Verifiable Sealed-Bid Auction on the Ethereum Blockchain

Hisham S. Galal and Amr M. Youssef

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

Abstract. The success of the Ethereum blockchain as a decentralized application platform with a distributed consensus protocol has made many organizations start to invest into running their business on top of it. Technically, the most impressive feature behind the success of Ethereum is its support for a Turing complete language.On the other hand, the inherent transparency and, consequently, the lack of privacy poses a great challenge for many financial applications. In this paper, we tackle this challenge and present a smart contract for a verifiable sealed-bid auction on the Ethereum blockchain. In a nutshell, initially, the bidders submit homomorphic commitments to their sealed-bids on the contract. Subsequently, they reveal their commitments secretly to the auctioneer via a public key encryption scheme. Then, according to the auction rules, the auctioneer determines and claims the winner of the auction. Finally, we utilize interactive zero-knowledge proof protocols between the smart contract and the auctioneer to verify the correctness of such a claim. The underlying protocol of the proposed smart contract is partially privacy-preserving. To be precise, no information about the losing bids is leaked to the bidders. We provide an analysis of the proposed protocol and the smart contract design, in addition to the estimated gas costs associated with the different transactions.

Keywords: Ethereum, Smart Contract, Sealed-Bid Auction.

1 Introduction

Online auctions have played an important role in the world economy by transferring trillions of dollars in exchange for goods and services in the recent decades. An auction is a platform for sellers to advertise the sale of arbitrary assets where buyers place competitive bids as the highest prices they are willing to pay. Practically, auctions promote many economic advantages for the efficient trade of goods and services. Traditionally, there are four main types of auctions [7]:

1. First-price sealed-bid auctions (FPSBA). Bidders submit their bids in sealed envelopes and hand them to the auctioneer. Subsequently, the auctioneer opens the envelopes to determine the bidder with the highest bid.
2. Second-price sealed-bid auctions (Vickrey auctions). It is similar to FPSBA with the exception that the winner pays the second highest bid instead.

3. Open ascending-bid auctions (English auctions). Bidders increasingly submit higher bids and stop bidding when they are not willing to pay more than the current highest bid.
4. Open descending-bid auctions (Dutch auctions). Auctioneer initially sets a high price, which is gradually decreased until a bidder decides to pay at the current price.

Arguably, the main advantage behind the sealed-bid auctions lies in the fact that no bidder learns any information about the other bids. Hence, the bidders are encouraged to bid according to their monetary valuation of the asset. However, a collusion between the auctioneer and a malicious bidder can break this advantage. In other words, there is a conflict between preserving the privacy of the bids and trusting the auctioneer to individually determine the winner. Hence, in online sealed-bid auctions, cryptographic protocols can be utilized to accomplish the publicly verifiable correctness without sacrificing the privacy of the bids.

According to a recent Reuters report [10], as part of the efforts to improve the transparency in government transactions, the Ukraine's justice ministry carried out trial auctions on top of the blockchain. The main goal is to make the auction system more transparent and secure such that the information is accessible to everyone to check if there is any manipulation or corruption.

Recently, cryptocurrencies have gained high popularity as evidenced by the surge in Bitcoin exchange rate. The foundation of cryptocurrencies is based on a decentralized public ledger on a peer-to-peer network that maintains the history of all transactions in an append-only fashion. Peers agree on the state of the ledger through an incentive-based consensus protocol. Additionally, cryptocurrencies also use cryptography to secure transactions as well as to control the creation of new currency units. Furthermore, many cryptocurrencies blockchains go beyond the simple means of payments. In fact, they provide a support for building and executing contracts on top of them. Simply, a smart contract is a piece of code that is stored and run on the blockchain. The smart contract resides passive until its execution is triggered by transactions. With the help of the consensus protocol, the contract is also guaranteed to be executed as its code dictates.

The Ethereum blockchain [17] presumably provides the highest support for smart contracts creation. Smart contracts are executed by a simple stack-based Turing complete 256-bit virtual machine known as the Ethereum Virtual Machine (EVM). Solidity is the common scripting language for writing smart contracts with a growing community. Ether represents the unit of currency in Ethereum and there are two types of accounts: externally owned accounts and contract accounts. An externally owned account is typically associated with a user, it consists of a unique public-private key pair. On the other hand, a contract account is controlled by the contract instead of a single private key. Transactions are created and signed by externally owned accounts. The receiver of the transaction can be an externally owned account or a contract account. In the former case, the transaction's purpose is to transfer ethers between users. Whereas in the latter case, the transaction triggers the execution of a function on the smart contract. Transactions also include a gas limit and a gas price; the amount of gas consumed to execute the transaction is converted into ethers using the gas price. These ethers are charged to the sender's account as transaction fees.

The Ethereum project has been planned in four locksteps [16]: Frontier, Homestead, Metropolis, and Serenity. Each update brings a set of approved Ethereum Improvement Proposals (EIP). Recently, the Ethereum blockchain has been upgraded to the first phase of Metropolis which is named Byzantium. The fork has been announced by the Ethereum team at the block number 4,370,000 [14]. Byzantium includes EIP-196 to efficiently perform elliptic curve point addition and scalar multiplication operations on *alt_bn128* curve [15]. Simply, it is a pre-compiled contract with a special address that is intercepted by the client software which provides an efficient native implementation for elliptic curve operations, rather than the inefficient EVM implementation. EIP-196 along with EIP-197 proposals prepare Ethereum for untraceable transactions by incorporating zk-SNARK similar to Zcash [12] blockchain.

Despite the flexibility and power of the smart contracts, the present form of the blockchain technologies lacks transactional privacy. Typically, every sequence of actions executed in the smart contract is propagated across the network and ends up being recorded on the blockchain. As a result, the lack of privacy is considered a major challenge towards the adoption of smart contracts as alternatives to many financial applications. Many individuals are not willing to reveal their financial transactions to the public. In this paper, we tackle this challenge and present an auction smart contract that utilizes a set of cryptographic primitives to guarantee the following attributes:

1. Bid privacy. All bidders cannot know the bids submitted by the others before committing to their own. This property is also guaranteed even in a collusion with a malicious auctioneer.
2. Posterior privacy. Given a semi-honest auctioneer, all committed bids are maintained private from the bidders and public users.
3. Bid Binding. Once the bid interval is closed, bidders cannot change their commitments.
4. Public verifiable correctness. The auction contract verifies the correctness of the auctioneer's work to determine the auctioneer winner.
5. Financial fairness. Bidders or auctioneer may attempt to deviate from the protocol and prematurely abort to affect the behavior of the auction protocol. The aborting parties are financially penalized while honest parties are refunded after a specific timeout.
6. Non-Interactivity. Bidders do not participate in complex interactions with the underlying protocol of the auction contract. In fact, no extra communications between the bidders and the auction contract are required aside from the submission of the bid commitments and the associated opening values.

We have also made our implementation prototype available on Git-hub [1] for researchers and community to review it.

[1] `https://github.com/HSG88/AuctionContract`

The rest of this paper is organized as follows. Section 2 provides a review of state-of-the-art research on auction solutions on the blockchain. The cryptographic primitives and the protocol for comparing the bids and verifying the correctness of the auction winner are presented in Section 3. In Section 4, we provide an analysis of the auction contract design and the estimated gas cost of the relevant transactions. Finally, we present our conclusions and future work in Section 5.

2 Related Work

Many of the previous research have focused on combining cryptocurrencies with secure multiparty computation protocols (MPC) and/or zero-knowledge proofs (ZKP). Typically, the cryptocurrency is used to incentive fairness and correctness, and avoid deviations from the MPC or ZKP protocol [1, 2, 8, 9]. Initially, each participant deposits an amount of cryptocurrency in a smart contract. These funds are reserved while the protocol is still running. Subsequently, once the protocol reaches a final state after an arbitrary timeout, the deposits get refunded only to the honest players. This in effect encourages parties to strictly follow the protocols to avoid the financial penalty.

Kosba et al. [6] presented Hawk, a framework for creating Ethereum smart contract that does not store financial transactions in the clear on the blockchain. One can easily write a Hawk program without having to implement any cryptography. The associated compiler utilizes different cryptographic primitives such as ZKP to automatically generate privacy-preserving smart contracts. A Hawk program contains public and private parts. The public part consists of the logic that does not deal with the data or the currency. Conversely, the private part is responsible for hiding the information about data and input currency units. The compiler translates the Hawk program into three pieces that define the cryptographic protocol between users, manager, and the blockchain nodes. The security of a Hawk program is guaranteed to satisfy *on-chain privacy* that protects the flow of money and data from the public view, and *contractual security* that protects the parties in the agreement of the contract from each other. Up to our knowledge, the Hawk framework has not been released yet on the project homepage `http://oblivm.com/hawk/download.html`.

Blass and Kerschbaum [3] presented *Strain*, a protocol to implement sealed-bid auctions on top of blockchains that protects the bid privacy against fully-malicious parties. To achieve efficiency and low latency cost, the authors avoided the use of highly interactive MPC primitives such as garbled circuits. Instead, they designed a two-party comparison mechanism executed between any pair of bidders in parallel. The outcome of the comparison is broadcasted to all bidders such that each one can verify it using ZKP. An additional ZKP protocol is used to verify that the comparisons only involved the committed bids. Moreover, to achieve fairness against prematurely aborting malicious parties, the protocol uses a reversible commitment scheme such that a group of bidders can jointly open the bid commitment. The authors mentioned that the proposed protocol leaks the order of bids similar to Order Preserving Encryption (OPE) schemes.

Snchez [13] proposed *Raziel*, a system that combines MPC and ZKP to guarantee the privacy, correctness and verifiability of smart contracts. The associated proofs of the smart contracts can effectively prove the functional correctness of a computation, besides to additional properties such as termination, security, pre-conditions and post-conditions. Furthermore, the author presented how a smart contract owner can prove its validity to third parties without revealing any information about the source code by using Zero-Knowledge Proofs to create Proof-Carrying Code certificates. Moreover, the author also proposed an incentive-based scheme for miners to generate preprocessed data of MPC.

3 Preliminaries

In this section, we briefly explain the cryptographic primitives that are utilized in the design of our proposed protocol:

1. Homomorphic commitment scheme that supports the addition operation on the underlying values
2. Zero-knowledge proof of interval membership $x \in [0, B]$.

3.1 Homomorphic Commitment Scheme

Our protocol makes an extensive use of Pedersen commitment scheme [11]. Let G and H be fixed public generators of the elliptic curve *alt_bn128* which is supported in EIP-196 and EIP-197 with the group order q [15]. The value of H is chosen such that neither the bidders nor the auctioneer know its discrete log. To commit a bid $x \in Z_q$, the bidder chooses a random $r \in Z_q$, then computes the commitment as $C = xG + rH$. Later, to open the commitment C, the bidder simply reveals the values of x and r. The Pedersen commitment scheme also possesses the homomorphic addition property on the underlying committed values by simply computing the point addition operation on the commitments. In other words, given two commitments $C_1 = x_1G+r_1H$ and $C_2 = x_2G+r_2H$, then $C_1+C_2 = (x_1+x_2)G+(r_1+r_2)H$ which is essentially the outcome commitment to $x_1 + x_2$.

3.2 Zero-Knowledge Proof of Interval Membership

We adapt the interval membership ZKP protocol proposed in [4]. Given an arbitrary number x which belongs to an interval $[0, B)$, the prover is able to convince the verifier that $x \in [-B, 2B)$. Since the financial values of bids cannot be negative numbers, the proved interval membership becomes $x \in [0, 2B)$. The protocol runs as follows:

1. **Commit.** The prover picks a number $w_1 \in [0, B]$ and sets $w_2 = w_1 - B$. Then, the prover sends the commitments $X = xG + uH, W_1 = w_1G + r_1H$, and $W_2 = w_2G + r_2H$ to the verifier.

2. **Challenge.** The verifier picks a random variable $b \in \{0, 1\}$.
3. **Response.** The prover sends one of the following responses to the verifier based on the value of b:
   – Case $b = 0$, the prover sends w_1, r_1, w_2, and r_2. The verifier checks $|w_1 - w_2| = B$, and the successful opening of the commitments W_1 and W_2.
   – Case $b = 1$, the prover sends $m = x + w_z$ and $n = u + r_z$, where $m \in [0, B)$ and $z \in \{0, 1\}$. The verifier checks $XW_z = (x + w_z)G + (u + r_z)H$.

In this protocol, the probability of cheating is $\frac{1}{2}$ which is non-negligible. However, with multiple k rounds of the protocol, the cheat probability becomes $\frac{1}{2^k}$.

3.3 Proving Claimed Inequality $x_1 > x_2$

Based on the primitives outlined above, we can prove that one bid is greater than another as follows. Suppose that $x_1, x_2 \in Z_q$, where q is a 256-bit prime number representing the order of *alt_bn128* elliptic curve as specified in EIP-197 and EIP-198 [15]. Then it is relatively easy to prove that $x_1 > x_2$ if and only if the following three interval membership hold (i) $x_1 \in [0, \frac{q}{2})$, (ii) $x_2 \in [0, \frac{q}{2})$, and (iii) $\Delta x_{1,2} \in [0, \frac{q}{2})$ where $\Delta x_{1,2} = (x_1 - x_2) \, mod \, q$.

In our work, the auctioneer acts as a prover and the auction contract acts as a verifier. Recall that in the interval membership ZKP, the prover is able to convince the verifier that $x \in [0, 2B)$ given that $x \in [0, B)$. As a result, we set an upper bound $V = \frac{q}{4}$ on the range of possible bids. Additionally, the auctioneer is not allowed to create any commitments for the bids, instead, the auctioneer only uses the commitments submitted by the bidders on the smart contract. The auction contract utilizes the additive homomorphic feature of Pedersen commitment scheme to compute the commitment to the differences between each pair of bids $\Delta X_{i,j} = X_i + (-1)X_j$.

4 Auction Smart Contract

In this section, we illustrate all the interactions between the bidders, the auctioneer, and the auction contract. Although our work applies to both types of sealed-bid auctions, we demonstrate the interactions in the case of FPSBA.

There are five sequential phases from the initial deployment of the auction contract to the collection of the highest bid from the winner given a successful verification of correctness. There are two methods to define phases of a smart contract: time interval and block interval. In time interval, the smart contract checks the time of the mined block (*block.timestamp or now*) which is specified by the block's miner. Ethereum developers discourage this method since it can be easily manipulated by the miners. On the other hand, in block interval, the smart contract loses the notion of time.

4.1 Phase 1: Contract Deployment and Parameters Setup

As shown in Fig. 1, the auctioneer initially deploys the auction contract on the Ethereum blockchain with the following set of parameters:

```
Create:   upon receiving from auctioneer A ($T_1, T_2, T_3, T_4, N, F, A_{pk}$) :
          Set state := INIT, bidders := {}, zkpCommits := {}
          Set highestBid := 0, winner := 0
          Set challengeBlockNumber := 0, challengedBidder := 0
          Assert $T < T_1 < T_2 < T_3 < T_4$
          Assert ledger[A] >= F
          Set ledger[A] := ledger[A] - F
          Set deposit := deposit + F
```

Fig. 1. Pseudocode for the deployment of the auction contract

1. T_1, T_2, T_3, T_4 define the time intervals for the following four phases: commitments of bids, opening the commitments, verification of the winner, and finalizing the auction, respectively.
2. F defines the amount of initial deposit of ethers received from the bidders and the auctioneer to achieve financial fairness against malicious parties.
3. N is the maximum number of bidders.
4. A_{pk} is the auctioneer's public key of an asymmetric encryption scheme.

4.2 Phase 2: Commitment of Bids

This phase starts immediately after the deployment of the auction contract. Each bidder submits a bid commitment using Pedersen commitment scheme along with the initial deposit F in ethers to the function Bid as shown in Fig. 2.

```
Bid:      upon receiving from a bidder B ($com_B$):
          Assert $T < T_1$
          Assert ledger[B] > F
          Set ledger[B] := ledger[B] - F
          Set deposit := deposit + F
          Set bidders[B].Commit := $com_B$
```

Fig. 2. Pseudocode for the Bid function

Suppose that an arbitrary bidder Bob is known to be very rich and is really interested in winning the auctioned item, i.e., Bob is very likely to be the one who submits the highest bid. Then, a collusion between a malicious bidder Alice and the auctioneer can eliminate Bob's winning chance by abusing the homomorphic property of the Pedersen commitment. The attack can be carried out as follows:

1. Bob submits the commitment $C_B = (xG + rH)$.

2. Subsequently, Alice submits the commitment $C_A = C_B + (G + H)$.
3. Bob reveals (x, r) to the auctioneer.
4. The auctioneer forwards (x, r) to Alice.
5. Alice reveals $(x + 1, r + 1)$.

To avoid this attack, we utilize Chaum-Pedersen non-interactive ZKP [5], which is not shown in Fig. 2. for the sake of simplicity. In this case, the above attack is not applicable because Bob sends commitments to random numbers rather than the actual bid which are subsequently challenged to verify the knowledge of values (x, r). As a result, Alice cannot succeed to imitate Bob's commitment since she will receive different challenges to verify the knowledge of $(x+1, r+1)$.

4.3 Phase 3: Opening the Commitments

Each bidder B_i sends the outcome ciphertext of encrypting (x_i, r_i) by the public key of the auctioneer A_{pk} to the function `Reveal` on the auction contract as shown in Fig. 3.

```
Reveal:    upon receiving from a bidder B (ciphertext):
           Assert $T_1 < T < T_2$
           Assert B $\in$ bidders
           Set bidders[B].Ciphertext := ciphertext
```

Fig. 3. Pseudocode for the Reveal function

The ciphertext are stored on the auction contract instead of being sent directly to the auctioneer in order to avoid the following attack scenario. Suppose a malicious auctioneer pretends that an arbitrary bidder Bob has not revealed the opening values of the associated commitment. In this case, Bob has no chance of denying this false claim. However, if the ciphertext are to be stored on the auction contract, then their mere existence successfully prevents this attack.

We have also taken into our account the possibility of the following attack as well. Suppose a malicious auctioneer intends to penalize an arbitrary bidder Bob by claiming that the decryption outcome of Bob's ciphertext CT_B does not successfully open Bob's commitment C_B. We prevent this attack by requiring the auctioneer to verify the opening correctness of the commitments once they are submitted by the bidders. In the case of unsuccessful opening, the auctioneer declares on the auction contract that the ciphertext associated with the bidder B is invalid. The honest bidder can deny this claim by revealing (x_B, r_B) to the auction contract. Subsequently, the auction contract encrypts the revealed values by the public key A_{pk}. If the outcome ciphertext is found to be equivalent to the previously submitted ciphertext, then the auction contract penalizes the auctioneer and terminates the auction after refunding the bidders. Otherwise, the bidder is penalized and the associated commitment is removed, such that only the valid commitments exist on the auction contract.

To guard against *forward search* attack on the submitted ciphertext, the parameter r in the opening values is a 256-bit random number that has no restriction on its value compared to the parameter x. Additionally, the opening values are combined to form one message which is passed to the encryption scheme.

4.4 Phase 4: Verification of Comparison Proofs

The auctioneer orders the bids to determine the wining bid x_w, the associated account address B_w and commitment C_w. Then, the auctioneer has to prove that $x_w > x_i$ for all $i \neq w$ and $0 < i < N$. The auction contract has a set of states to impose an order on the functions being invoked by the auctioneer for verification. Initially, the auctioneer calls the function `ClaimWinner` to claim that a winner is found by specifying the account address and opening values of the bid commitment as shown in Fig. 4.

```
ClaimWinner: upon receiving from auctioneer A ($B_w$, $x_w$, $r_w$):
    Assert state = INIT
    Assert $T_2 < T < T_3$
    Assert $x_w < V$
    Assert $B_w \in$ bidders
    Assert bidders[$B_w$].commit = Pedersen.Commit($x_w$, $r_w$)
    Set winner := $B_w$
    Set highestBid := $x_w$
    Set state := Challenge
```

Fig. 4. Pseudocode for the ClaimWinner function

Recall that the interval membership ZKP has a probability of cheating $\frac{1}{2}$ which is non-negligible; however, this probability can be further reduced to $(\frac{1}{2})^k$ by running the protocol k times. Moreover, in the *challenge* step, the verifier sends to the prover a random value $b \in \{0, 1\}$ which has to be non-predictable. However, smart contracts cannot send data to externally owned accounts, (i.e., the auction contract cannot send a challenge value to the auctioneer). Hence, we utilize a non-interactive interval membership ZKP to prove $x_i \in [0, \frac{q}{2})$ as follows:

1. **Commit:** The auctioneer chooses k-pairs of $(w_{1,j}, w_{2,j})$ where $w_{1,j} \in [-V, V)$ and $w_{2,j} = w_{1,j} - V$ such that $|w_{1,j} - w_{2,j}| = V$ for $1 \leq j \leq k$. Then, the auctioneer invokes the function `ZKPCommit` with the account address of the challenged bidder and the commitments to w_1 and w_2 as shown in Fig. 5.

```
ZKPCommit: upon receiving from auctioneer A ($B_i, commits$):
           Assert state = Challenge
           Assert $T_2 < T < T_3$
           Assert $B_i \in$ bidders
           Set zkpCommits :=$commits$
           Set challengeBidder := $B_i$
           Set challengeBlockNumber := QueryBlockNumber()
           Set State := Verify
```

Fig. 5. Pseudocode for the ZKPCommit function

2. **Challenge and Response:**
   - The auctioneer receives a transaction receipt which includes the hash of the block containing the transaction after it has been confirmed. The `ZKPCommit` function has no access to this hash while it is being executed; therefore it stores the current block number in `challengeBlockNumber`.
   - The least significant k-bits of the hash are chosen as the challenge b_j.
   - The auctioneer creates k responses R_j based on the values of b_j.
   - Case $b_j = 0$, then $R_j = \{w_{1,j}, r_{1,j}, w_{2,j}, r_{2,j}\}$.
   - Case $b_j = 1$, then $R_j = \{m_j, n_j, z\}$ where $m_j = x_j + w_{z,j}$, $n_j = u_j + r_{z,j}$ such that $m_j \in [0, V)$ and $z \in \{1, 2\}$.
   - The auctioneer invokes the function `ZKPVerify` with input parameter *responses* which is an array of R_j as shown in Fig. 6.

```
ZKPVerify: upon receiving from auctioneer A ($responses$)
           Assert State = Verify
           Assert $T_2 < T < T_3$
           Set hash := QueryBlockHash(challengeBlockNumber)
           for $j \in [1, k], R_j \in responses, C_j \in zkpCommits$
               Set $b_j$ := Bit(hash,$j$)
               if $b_j = 0$
                   Assert VerifyFirstCase($C_j, R_j$)
               else
                   Assert VerfiySecondCase($C_j, R_j$)
           Set bidders[challengeBidder].ValidBid := true
           Set state := Challenge
```

Fig. 6. Pseudocode for the ZKPVerify function

As explained in Section 3, three interval membership ZKP are required to prove that $x_w > x_i$. However, since the bid of the winner B_w is revealed, then the number of proofs is reduced to two. In other words, the auctioneer has to prove the interval membership for all bids x_i other than the winning bid and their associated differences Δ_{wi}. The function `ZKPCommit` and `ZKPVerify` contain extra logic to also verify the correctness of $\Delta_{wi} \in [0, \frac{q}{4})$.

4.5 Phase 5: Finalizing the Auction

After the successful verification of correctness, the auctioneer invokes the function `VerifyAll` as shown in Fig. 7 to change the state of the auction contract so that the winner can pay the winning bid.

```
VerifyAll    upon receiving from auctioneer A ()
             Assert state = Challenge
             Assert $T_2 < T < T_3$
             For all b $\in$ bidders - $\{winner\}$
                 Assert b.ValidBid = true and b.ValidDelta = true
             Set State := ValidWinner
```

Fig. 7. Pseudocode for the VerifyAll function

Subsequently, The winner invokes the function `WinnerPay` to deposit the difference between the winning bid and the initial deposit F as shown in Fig. 8.

```
WinnerPay    upon receiving from a bidder B ("$winnerPay$")
             Assert State = ValidWinner
             Assert $T_3 < T < T_4$
             Assert B = winner
             Assert ledger[B] > highestBid - F
             Set ledger[B] := ledger[B] - highestBid +F
             Set deposit := deposit + highestBid - F
             Set state := WinnerPaid
```

Fig. 8. Pseudocode for the WinnerPay function

The auction contract guarantees to refund the initial deposit to all honest players after the time T_3 as shown in Fig. 9. In the case of invalid proofs, it penalizes the auctioneer and refunds all bidders. Otherwise, it refunds the losing bidders and the auctioneer as well. It is also clear that the only way for the winner to refund the initial deposit is by invoking WinnerPay function.

```
Timer
             if $T > T_3$ then
                  if state $\neq ValidProof$ then
                      refund(F) for all b $\in$ bidders
                  else
                      refund(F) to auctioneer A
                      refund(F) for all b $\in$ bidders - $\{winner\}$
```

Fig. 10. Pseudocode for the Timer function

4.6 Gas Cost

We have created a local private Ethereum blockchain to test our prototype using the *Geth* client version 1.7.2. To support the Byzantium EIP-196 and EIP-197, the *genesis.json* file has to contain the attribute {*"byzantiumBlock": 0*}. Additionally, since Ethereum does not support timer triggered functions, we have implemented a *Withdraw* function that is invoked by an explicit request from the honest players to refund their initial fairness deposit. We have tested the auction contract with ten bidders, and we have set $k = 10$ as the number of multiple rounds to verify interval membership NiZKP which results in a probability of cheat less than 0.001. The upper bound on bid values is up to 250-bit length which is very adequate for financial values. The Pedersen commitment size is 512-bits that represent two points on the elliptic curve. The ciphertext submitted to the `Reveal` function is 1024-bits. Table 1 shows the consumed gas and the equivalent monetary cost in *US* dollars for invoking different functions on the auction contract. As of November 30, 2017, the ether exchange rate is 1 ether = 450\$ and the gas price is approximately 20 *Gwei* $= 20 \times 10^{-9}$ ether. Furthermore, the execution of "heavy" functions in Ethereum is not only costly in dollar terms, but may be even impossible, if the function's gas requirements exceed the block gas limit. The block gas limit at time of writing is 8m gas, whereas the most expensive protocol function consumes 2m gas, which seems feasible

Table 1. Consumed gas cost for different functions of the Auction contract

Function	Gas units	Gas cost (USD)
Deployment	3131261	28.18
Bid	130084	1.17
Reveal	132849	1.19
ClaimWinner	166288	1.49
ZKPCommit	656689	5.91
ZKPVerify	2002490	18.02
VerifyAll	46580	0.42
Withdraw	47112	0.42

5 Conclusion and Future Work

In this paper, we presented a smart contract for a verifiable sealed-bid auction on the Ethereum blockchain. We utilized Pedersen commitment scheme along with ZKP of interval membership to create the underlying protocol. The auction contract maintains the privacy of bids such that bidders do not learn any information about the other bids when they commit. Additionally, the auction contract also exhibits the public verifiable correctness as it is designed to verify the proofs claimed by the auctioneer to determine the winner. Moreover, no complex interaction is required from the bidders other than submitting and revealing the commitments to their bids. The proposed protocol can be easily modified to support the full privacy of all bids including the winner's bid if there is a desire to receive the payment of winning bid aside from the blockchain. For future work, we will investigate other approaches applicable to the Ethereum blockchain where we can also protect the privacy of bids from all parties including the auctioneer.

References

1. Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy (SP)*, pages 443–458. IEEE, 2014.
2. Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *International Cryptology Conference*, pages 421–439. Springer, 2014.
3. Erik-Oliver Blass and Florian Kerschbaum. Strain: A secure auction for blockchains. Cryptology ePrint Archive, Report 2017/1044, 2017. `https://eprint.iacr.org/2017/1044`.
4. Ernest F Brickell, David Chaum, Ivan B Damgård, and Jeroen van de Graaf. Gradual and verifiable release of a secret. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 156–166. Springer, 1987.
5. David Chaum and Torben P Pedersen. Wallet databases with observers. In *Crypto*, volume 92, pages 89–105. Springer, 1992.
6. Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 839–858. IEEE, 2016.
7. Vijay Krishna. *Auction theory*. Academic press, 2009.
8. Ranjit Kumaresan and Iddo Bentov. Amortizing secure computation with penalties. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 418–429. ACM, 2016.
9. Ranjit Kumaresan, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. Improvements to secure computation with penalties. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 406–417. ACM, 2016.
10. Alessandra Prentice Olena Vasina. Ukrainian ministry carries out first blockchain transactions. Reuters Technology News. `https://goo.gl/J8X1up`.
11. Torben Pedersen and Bent Petersen. Explaining gradually increasing resource commitment to a foreign market. *International business review*, 7(5):483–501, 1998.
12. Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy (SP)*, pages 459–474. IEEE, 2014.
13. David Cerezo Snchez. Raziel: Private and verifiable smart contracts on blockchains. Cryptology ePrint Archive, Report 2017/878, 2017. `https://eprint.iacr.org/2017/878`.

14. Ethereum Project Team. Byzantium hf announcement, 2017. https://blog.ethereum.org/2017/10/12/byzantium-hf-announcement/.
15. Ethereum Project Team. Ethereum improvement proposals, 2017. https://github.com/ethereum/EIPs.
16. Ethereum Project Team. The ethereum launch process, 2017. https://blog.ethereum.org/2015/03/03/ethereum-launch-process/.
17. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151, 2014.