

Ethereum Based Sealed Bid Auction for ERC20 Tokens

Aashish Arora, Anisha Kumar, Chitrarth Patel and Ridhi
Electrical and Computer Engineering Department, University of Victoria

Abstract

Keywords: Blockchain, Ethereum, Blind Auction, ERC-20 tokens

Ethereum smart contracts have revolutionized the way to do any sort of processing in a decentralized manner. Starting from financial services, to fundraising, supply chain management, KYC (Know your customer), securitization of assets and even gaming, smart contracts on Ethereum have found their place everywhere. Another important application of Ethereum smart contract is using them to create virtual Marketplaces including exchanges. We have used Ethereum smart contract here to create a Sealed-Bid platform wherein the smart contract itself functions as the auctioneer. We have built our own token using the ERC-20 standard and it is the commodity we wish to sell during the auction. The winner of the auction may purchase other items using the token, depending on who accepts it for an exchange of the item; or store it as an asset. The token owner runs the smart contract and multiple bidders submit their encrypted bids to the contract during the bidding period. In the revealing phase, bidders reveal their original bids, which are compared with the encrypted bids they submitted. If both matches, the bidders can reveal the winner in the next phase. Bidders who have lost can claim their amount back in the withdrawal period. There are numerous advantages of using blockchain for a sealed-bid auction which arise from the fact that blockchain is decentralized and cannot be tampered with. We have provided the smart contract design for creating the token to be used during the auction as well as the smart contract for the auction itself.

1 Introduction

An auction can be described as process of selling goods or services by placing them for bid, asking audience to bid, and then selling the item to the highest bidder (or in some cases, the second highest bidder depending on the auction type). Three entities are involved in an auction – the owner of the commodity or service being sold, the auctioneer and the bidders who wish to buy the product or service. There are several types of auction depending upon what/who drives the auction – time, or bidders and vary in the number and type of participants. The type of auction for which we have developed the smart contract design here is called Sealed-First-Price Auction or blind auction.

Blind Auction, also known as first-price sealed-bid auction (FPSB), the base price of the commodity is set and all bidders are allowed to bid simultaneously but no participant knows the bid the other, in the essence that it is sealed. The bids of all participants are compared, and the commodity goes to the highest bidder. This type of auction focuses on the valuation of the bidder. All bidders have a certain valuation, and no bidder will bid more than that valuation as that will make them lose their net value. If the bidder pays exactly equal to their value, they will not suffer any loss on winning the auction but will also not gain any positive value. On the other hand, if they bid a little less than their value, the winner may have some profit/gain depending on what the next highest bids were.

The concept of value explained above is used in the smart contract design as stake value. Each bidder has a certain stake value that limits them from bidding an immensely large bid and winning but never showing up for the payment. The stake value, as explained will always be a little higher than the bid amount and must be sent during the auction to the auctioneer, here our service smart contract. Once the winner has been decided, the bidders who lost can claim the whole stake amount back and the winner can claim the

difference between the stake value and the bid value back. There are multiple important steps in between that ensure the security of the processes mentioned above. We have made sure that no bidder can claim more than what he paid during the bidding period as stake value, and that there are no false bidders trying to sabotage the system. The commodity being sold in the blind-auction is a token that we have created using the ERC-20 standard as explained further.

2 ERC-20 Tokens

Applications built on Ether generally use a token to function as the currency inside the application and the developers of the application are charged in Ether for being able to deploy and run applications using the Ethereum blockchain. This can be thought of as a poker table, where in the players (or the users of the application) deal only in the tokens assigned by the table owner, whereas the tokens can be bought using actual money (or Ether). These tokens are created by smart contracts itself, and then are used by the service contract built by the developers by importing them. ERC-20 is a universal language that generally all tokens created on the Ethereum blockchain network utilize. This standard was created so as to maintain uniformity in writing, understanding and deploying tokens easily without any hassle of translation. ERC-20 defines a set of functions which can be defined in the smart contract of the token and can be called in the service contract depending on the usage and functionality of the application. It has some optional and some mandatory parameters defined. These are listed as follows-

- Optional
 - Token Name
 - Symbol
 - Decimal (up to 18)
- Mandatory
 - totalSupply
 - balanceOf
 - transfer
 - transferFrom
 - approve
 - allowance

For the token we have created, we have just used the functions – Token Name and transfer, as they're the only two required by the application for functioning. The Token Name function is simply for defining the name of the token (in our case – Auction Token) and the function Transfer is to be able to transfer balance

from the token contract to the winner of the auction, explained later.

ERC20 tokens can be considered as assets on the Ethereum blockchain, which can be stored, sent or received. They work differently than other cryptocurrencies i.e. Bitcoin, Ether as tokens don't have their blockchain, instead tokens work on the top of Ethereum blockchain i.e. on smart contracts are used to create tokens. Tokens can be sent using transactions and stored in Ethereum accounts. In this paper, ERC20 tokens are used as a good to be auctioned. At the end of revealing period, the tokens will be given to highest bidder.

3 Smart Contract for Tokens

To handle the initial offering and transfer of tokens, we first developed an ERC20 compliant token contract named 'Auction Token', which has all the necessary functions to maintain direct and indirect token transfers. The totalSupply is the total value of the tokens that are created and are sent to the auction owner. This value is set to the initial supply set by the auction owner, which is also his balance. The function transfer is used to indicate the address to which the tokens have to be sent along with the value as shown below-

function transfer (address _to, uint256 _value)

It returns a Boolean value depending on if it was successful or not. This function is called by the owner of the auction at the end of identifying the winner, when the owner attempts to transfer the commodity – the Auction Token to the highest bidder. The security checks required while calling this function are – checking if the balance of the owner's account is greater than or equal to the amount of token to be sent and that the amount is not sent to the null address. If the amount is sent to the null address, the tokens will be burnt, and the owner will lose them. The service contract will specifically call functions from the token contract, which can be seen in the pseudocode explained below.

4 Smart Contracts

In our project of building an online sealed bid auction, we are making a smart contract which is like a self-enforcing piece of code managed by peer to peer network of computers. It has a set of rules, which when agreed upon, allow the participants to move forward and have an interaction or a deal with each other, like an 'if-then' principle in coding, we move ahead only if certain conditions are met which in this case are a set of rules. It helps builds the trust of the parties involved to carry out online transactions which otherwise is a primary concern in the world of internet. Since, there

are multiple participants involved, and the specifics of the transaction, like the amount, the stakes are stored in a decentralized blockchain network, and also owing to the transparent nature of the transaction, trust is guaranteed. Smart contracts also take out the need of an inter-mediator to carry out transactions, they're autonomous, so it rules out any mediator fee.

As discussed earlier, smart contracts are unlike your ordinary contracts are codes that are created by users to perform certain actions and Ethereum provides a platform to run and execute these contracts. First deployed by Bitcoins, these supported transactions by validating mechanisms used by nodes to transfer currency from one account to another, but it had certain restrictions that are overcome by Ethereum, which unlike its counterpart enables its users to write their own code and execute them to perform a specific command or transaction, it provides fluidity. It effectively controls and manages transactions of tokens and tokenized assets

5 Solidity

Now, we have a complete separate language to write and execute these smart contracts, called Solidity which is a high-level object-oriented programming language, based on C++, Python, and JavaScript. Since, the developers are writing their own code, they get to modify and improvise the commands as to how they want the transaction to proceed, and Solidity provides them with just the right platform. The Ethereum blockchain is decentralized, i.e., the smart contract acts as auctioneer, and it is immutable, so its secure and makes it hard for anyone to tamper with the data, which is also made possible by the use of state-of-the-art cryptography making it an even more secure network.

Smart contracts run in an isolated environment, oblivious to what other smart contracts are performing. This run-time environment is called EVM or Ethereum Virtual Machine that has no access to other networks or contracts. Once we have created a transaction, we have to pay a gas amount to deploy it which is needed to limit the work required to execute the transaction while also paying to execute the transaction, this is usually a very minimal amount. Solidity also provides a specially indexed data structure to store data in an efficient and cryptographically secure manner, this is called Logs. Logs are related to the address of the smart contracts and are incorporated into the blockchain.

6 Pseudo Code

1. Contract Deployment

- The contract in our case will be deployed by the auction owner and he will pay the gas fees for hosting the auction. Firstly, the contract deployment function works as below:
- Constructor:
 - from the auction owner `Bauction`
 - Set Bidding Phase := `BiddingTimeLimit`
 - Set Revealing Phase := `RevealingTimeLimit`
 - Set Base Price := `basePrice`
 - Set Auction token address := `_token`

2. Bidding Period

- Start Bidding:
 - from the Bidder
 - Requires := current time j bidding phase
 - Call `makeSeal` function := `keccak256(BidAmount + nonce)`
 - Call `makeBid` function := Insert 256 bit bid hash
 - Assert Stake Value := Amount of stake

3. Revealing Period

- Start Revealing:
 - from the Bidder
 - Requires := bidding phase j current time i revealing phase
 - Call `RevealWinner` function := `keccak256(BidAmount + nonce)`
 - Compare if hash matches that of bidder
 - Requires := bid value j = stake value
 - Sets the highest bidder, cuts bid value from stake

4. Revealing Period End

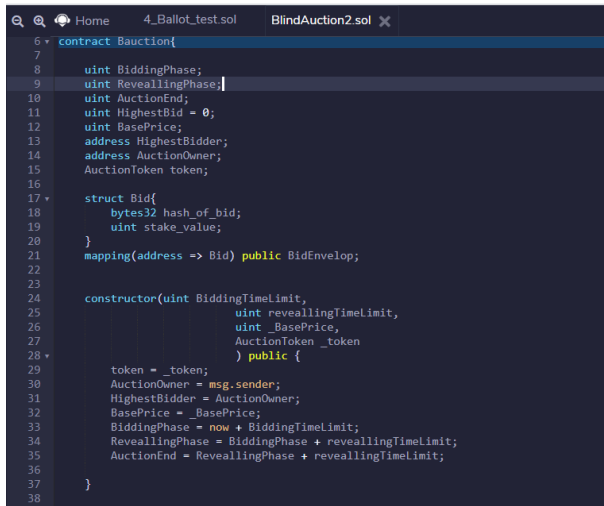
- Winner/ End auction:
 - from the Auction Owner
 - Call `get_amount` function := transfer highest bid to auction owner
 - Call `ResetAuction` function := called by auction owner, resets all variables
 - from the bidder
 - Requires := current time i revealing phase
 - Call `withdraw_stake_amount` function := return stake to bidder's account
 - given bidder != highest_bidder
 - Call `getWinner` function := returns Highest Bidder

- Call `getWinningPrice` function := returns Highest Bid
- Call `getStakeOfBidder` function := returns stake of each bidder
- Call `get_tokens` function := transfer tokens to highest bidder
- Call `get_bal` function := returns the number of tokens transferred

7 Explanation

A contract in Solidity is a collection of code with all its functions and data, called state in the case of Solidity, that is stored at specific location on the Ethereum Blockchain.

Below are the snippets from our code:



```

6 contract Bauction{
7
8     uint BiddingPhase;
9     uint RevealingPhase;
10    uint AuctionEnd;
11    uint HighestBid = 0;
12    uint BasePrice;
13    address HighestBidder;
14    address AuctionOwner;
15    AuctionToken token;
16
17    struct Bid{
18        bytes32 hash_of_bid;
19        uint stake_value;
20    }
21    mapping(address => Bid) public BidEnvelop;
22
23
24    constructor(uint BiddingTimeLimit,
25                uint revealingTimeLimit,
26                uint BasePrice,
27                AuctionToken _token
28                ) public {
29        token = _token;
30        AuctionOwner = msg.sender;
31        HighestBidder = AuctionOwner;
32        BasePrice = _BasePrice;
33        BiddingPhase = now + BiddingTimeLimit;
34        RevealingPhase = BiddingPhase + revealingTimeLimit;
35        AuctionEnd = RevealingPhase + revealingTimeLimit;
36
37    }
38

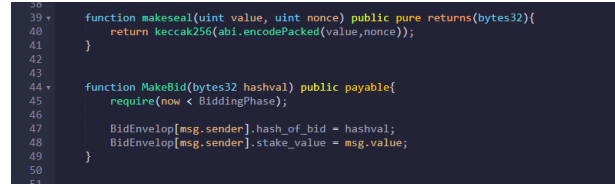
```

Figure 1: Begin Contract: Bauction

We created a contract called Bauction, it declares the state variables as the BiddingPhase, RevealingPhase, AuctionEnd, HighestBid = 0, BasePrice with uint as the type(unsigned integer of 256 bits), HighestBidder, AuctionOwner with address as the type(160-bit value for storing address) and AuctionToken token to set the address of the auction token.

We have a structure named Bid that stores the values of the stake with the hash of the bid, next we map this structure address with the BidEnvelop to store the hash and stake amount associated with each bidder at their respective addresses. We also have the constructor that clearly sets the values for the state variables.

When the bidding phase starts, each bidder calls the above described two function, i.e., `makeSeal` and `MakeBid`. The first function takes the bidding amount and the nonce, encodes them together into a single value and generates the hash using the `keccak256` function, later after the hash is generated, the former function is



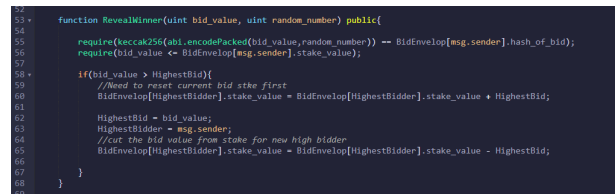
```

39
40 function makeSeal(uint value, uint nonce) public pure returns(bytes32){
41     return keccak256(abi.encodePacked(value,nonce));
42 }
43
44 function MakeBid(bytes32 hashval) public payable{
45     require(now < BiddingPhase);
46
47     BidEnvelop[msg.sender].hash_of_bid = hashval;
48     BidEnvelop[msg.sender].stake_value = msg.value;
49 }
50
51

```

Figure 2: Bidding Phase Begins

called which requires that the current time t Bidding-Phase. It takes two inputs, the hash generated by the `makeSeal` function as well as the stake amount(which ensures the authenticity of the bidder and protection from frauds) and stores them in the `BidEnvelop` which associates them with individual bidders address.



```

32
33 function RevealWinner(uint bid_value, uint random_number) public{
34
35     require(keccak256(abi.encodePacked(bid_value,random_number)) == BidEnvelop[msg.sender].hash_of_bid);
36     require(bid_value <= BidEnvelop[msg.sender].stake_value);
37
38     if(bid_value > HighestBid){
39         //Need to reset current bid the first
40         BidEnvelop[HighestBidder].stake_value = BidEnvelop[HighestBidder].stake_value + HighestBid;
41
42         HighestBid = bid_value;
43         HighestBidder = msg.sender;
44         //cut the bid value from stake for new high bidder
45         BidEnvelop[HighestBidder].stake_value = BidEnvelop[HighestBidder].stake_value - HighestBid;
46
47     }
48 }
49

```

Figure 3: Revealing Phase Begins

After the bidding period ends, the revealing phase begins and the `RevealWinner` function is called by each bidder. This function takes 2 inputs, the bid value and the nonce, generates their hash and compares it with the previously generated hash of the said bidder, after which it checks whether the bid_value \leq stake_value. If so, it sets the `HighestBid` and the `HighestBidder` by comparing the bid values of all the variables and stores them in the state variable and also updates the stake value simultaneously for the highest bidder as well as the previous bidder with lower bid value.

Once, the revealing period is over each bidder can view who the highest bidder was, and what their winning bid was by calling `getWinner` and `getWinningPrice` functions respectively. To view the stake of each bidder, `getStakeOfBidder` function is used, and `get_bal` function is used to reveal the winning number of tokens. They can call `withdraw_stake_amount` function to withdraw their stake, given current time t RevealingPhase and current bidder $! =$ HighestBidder, the functions sets the stake_amount as 0 in the `BidEnvelop` and transfers that value to the bidder's account.

The winner of the bid can retrieve his tokens by calling the function `get_tokens`. After the auction ends, the owner calls the function `get_amount` to transfer the Highest Bid amount to his account and resets the state variables to default values by calling the `ResetAuction` function.

```

71 function withdraw_stake_amount() public{
72     require(now > ReveallingPhase);
73     require(msg.sender != HighestBidder);
74     uint stake_amount = BidEnvelop[msg.sender].stake_value;
75     require(stake_amount > 0);
76     BidEnvelop[msg.sender].stake_value = 0;
77     msg.sender.transfer(stake_amount);
78 }
79
80 //for owner of the auction
81 function get_amount() public{
82     require(now > ReveallingPhase);
83     require(msg.sender == AuctionOwner);
84     msg.sender.transfer(HighestBid);
85 }
86
87 function get_tokens() public{
88     require(now > ReveallingPhase);
89     require(msg.sender == HighestBidder);
90     uint bal = token.balanceOf(this);
91     token.transfer(HighestBidder, bal);
92 }
93
94 function ResetAuction() public {
95     require(msg.sender == AuctionOwner);
96     require(now > AuctionEnd);
97     HighestBid = 0;
98     HighestBidder = address(0);
99     delete BiddingPhase;
100    delete ReveallingPhase;
101    delete AuctionEnd;
102 }

```

Figure 4: Revealing Phase Begins

8 Conclusion

In this paper, we proposed an implementation of Sealed Envelope Auction using Ethereum Smart Contracts. Ethereum being very flexible and scalable blockchain reserves plenty of advantages for these types of applications under smart contracts. Smart Contracts are immutable, transparent and self-verifiable, thus provides security, trust and dynamicity to the applications. We used ERC-20 tokens for the auction owner to ask the bidders to bid upon. On deploying the contract for the auction, the auction owner needs to provide the tokens to the contract and the contract stores it for the winner. The bidding period allows the bidders to bid on the tokens, provide hash and stake value for the bid. As the bid is a hash, it will be hidden from the world. After revealing their bids in the revealing period, the bidders can check the winner and the winning amount. If selected as the winner, the bidder will get the extra stake value back and the tokens that were on the auction from the contract. The proposed code can be modified as per the needs and demand of the auction. The code can be constructed in such a manner that it can be hosted on a private URL to keep the auction hidden from the world. Thus, Ethereum Blockchain provides very interesting platform for the new technological world where privacy and security matters a lot.

9 References

1. Ethereum 101. (2017, March 30). Retrieved from Coindesk: <https://www.coindesk.com/learn/ethereum->

101/what-is-ethereum

2. Introduction to Smart Contracts. (n.d.). Retrieved from Solidity Documentation: <https://solidity.readthedocs.io/en/v0.5.10/introduction-to-smart-contracts.html>
3. Metcalfe, W. (2020, April 16). Ethereum, Smart Contracts, DApps. Retrieved from Springer: https://link.springer.com/chapter/10.1007/978-981-15-3376-1_5 Solidity. (n.d.). Retrieved from Solidity Documentation: <https://solidity.readthedocs.io/en/v0.7.0/>
4. Voshmgir, S. (2019, July). Smart Contracts. Retrieved from Blockchainhub Berlin: <http://blockchainhub.net/smart-contracts/>
5. What Are Smart Contracts? Guide For Beginners. (n.d.). Retrieved from Cointelegraph: <https://cointelegraph.com/ethereum-for-beginners/what-are-smart-contracts-guide-for-beginners>
6. William, M. (2018, May 12). ERC-20 Tokens, Explained. Retrieved from Cointelegraph: <https://cointelegraph.com/explained/erc-20-tokens-explained>
7. Youssef, H. S. (n.d.). Verifiable Sealed-Bid Auction on the Ethereum. Quebec, Canada: Concordia Institute for Information Systems Engineering