ECE 579: Blockchain and Cryptocurrencies 2020

Assignment 4

In this assignment you need to implement an auction application using Smart Contracts. You need to use Solidity language for writing the Smart Contract. You can use Remix website for contract deployment and tests: http://remix.ethereum.org. It is easier than creating a test network on your PC with Geth.

In your implementation you need activate and use "Oraclize" application to calculate an exchange rate between USD and Ethereum. Also, floating- and fixed-point arithmetic is not fully supported in Solidity. You may declare exchange as an integer and round the numbers while computing division.

Note: Oraclize has some limitation in terms of number of trails that you can perform in a day. In order to overcome this issue you can create a function to enter the exchange rate yourself. You can add the "Oracilze" capability later.

Design Overview:

We have two types of users; Auctioneer or Auction Authority.

Auctioneer: The person who enters an auction and makes a bid. They will be EOA type of accounts in Ethereum.

Auction Authority: The authority is a single account and it is an EOA account. The account will deploy Smart Contract for the Auction Protocol. Auction process will be done in steps, and most of the steps will be executed (calling functions) by the Auction Authority.

Auction Authority Smart Contract:

Smart Contract needs to do the following:

Step 1:

- It registers any given public key address as a bidder. The Auctioneers can register only once.
- Once enough users are registered, Auction Authority closes the registration.

Step 2:

- Each Auctioneer is allowed to bid only once. In order to avoid other Auctioneers to peak into other bids, an Auctioneer bids by commitment. This means when Auctioneers are bidding, they have to create a message *message_bid* = (*bid in USD, random seed, public key*). Later, Auctioneer needs to take the hash of the *message_bid*: H(*message_bid*). This way, Auctioneer will commit a value and send it to the Auction Authority and avoid other Auctioneers' to see his bid. (There are default hash functions available in Solidity)
- Each user will send their commitment to the Smart Contract with enough Ethereum (value section in Ethereum Transaction). Usually, "value" should be larger than the bid itself.
- Once all Auctioneers committed their bids, the Auction Authority stops the bidding process.

Step 3:

- After Auctioneers commit their bids, they send *message_bid* = (*bid in USD, random seed, public key*) to the Smart Contract. Smart Contract checks if each commitment is true or not: (H(message_bid)_stored == H(message_bid)_recieved). If commitment is true, it means that Smart Contract received the correct bidding value. If it is not true, it discards the bid for that user and holds all Ethereum ("value") send by that user as a penalty for trying to deceive the system.

Step 4:

- Once each users' bid is collected, using Oraclize Application, we check the exchange rate for Etherum/USD.
- Among the bids that are valid, we take the largest bid as the winner of the auction as long as bidder's "value" is larger than the bid itself. If it is not, it means he bided coins that he does not have. Therefore, Smart Contract will seize the coins and check for the second largest bid if it is the winner. The process continues until we find the largest bidder that sent "value" larger than the bid. (In order to compute the coin amount, you need to use the exchange rate you learned by the Oraclize Application.)
- Once the auction is finalized, all valid Auctioneers' bids will be returned to the original owners accept the winner and invalid bidders.

Functions of Smart Contract:

Functions that you need on your Smart Contract (you may add as needed):

- **RegisterAuctioneer ():** Register Auctioneer by using the address of an Auctioneer. This is going to be executed by Auctioneers. You hold the Auctioneer addresses on a list. (You may use msg.sender to register the address of the Autioneer)(Create a variable: address[] auctioneer)

- **RegisterStop():** Stop registration. (Should be executed by only Auction Authority)

- **CommitBid(bytes32 hash_commit):** Stores the commitment and "value" send by the auctioneer (Used by Auctioneers. Each Auctioneer is allowed to submit only their bids and only once). (Create a variable: mapping(address => byte32) public commitment)

- **CommitBidStop():** Stop further commitments. (Only by the Auction Authority)

- **SendBid(random seed, bid in USD):** Take the inputs and check if the commitment matches H(message_bid)_stored = H(bid in USD, random seed, public key). If the commitment matches, the bid is valid. If the commitment does not match, it means the bidder changed the bid, so the Auction Authority can punish the Auctioneer by holding into the value send by the auctioneer. (Used by Auctioneers. Each Auctioneer is allowed to submit only their bids and only once)

- **ComputeExchange():** Take the exchange rate USD/Ethereum using the Oraclize API (Only by Auction Authority). (Create a variable: uint256)

- **ComputeWinner():** Compute the winner among the valid bids. If a bidder's bid is larger than the value that is send (bid>value), it means it did not put enough coins for the bid. The authority should check the next largest bidder and hold onto the coins that are sent by the first one. (Only by Auction Authority)

  Once the winner is calculated, all valid Auctioneers' bids will be returned to Auctioneers. Furthermore, winner Auctioneer will receive the overhead = "value" – bid from the Auction Authority. Auctioneers that tried to cheat during the bid process will never receive back their coins. Also, the winner's address should be added to a winner list (WinnersList) so that we can keep track of the winners for any auction. (Create a variable: mapping(address => byte32) public winners)

- **ResetAuction():** It will reset all the auction after an auction completed. You may need to clean the lists, arrays and any other variable that is defined and used along the process. Reset <u>should not clear</u> the WinnersList. (Only by the Auction Authority)

Second Smart Contract for Hash Calculation:

During the commitment, you need to compute the hash of the bid. In order to do that, you may deploy a smart contract that takes *message_bid* = (*bid in USD*, *random seed*, *public key*) as an input, computes and returns the result.

Demo and report requirements:

Once you showed your demo, please write a short report that explains what you implemented. You can include screenshots that shows the cases you presented in your demo.

ECE579 Assignment 4
Sign-off Sheet

Student 1: ____________________________________ **ECE mailbox:** ___________________

Student 2: ____________________________________ **ECE mailbox:** ___________________

Registration (Only once per Auctioneer)	5	
Close registration (Auction Authority only)	5	
Auctioneers can only bid once	5	
Auctioneer bids by commitment (only Auctioneer can commit its bid)	10	
Auctioneers commit their bids and Auction Authority (AA) stops the bidding process (only by AA)	5	
Auctioneers send the committed messages and AA checks if it matches with previous commitment (if it does not, discard the bid)	25	
Get exchange using Oraclize App	10	
AA calculates the winner (completes necessary checks such as value>bid)	10	
AA finalize the bid by returning the money except winners and cheaters (value<bid)	15	
Add the winner to the winners list	5	
Reset Auction	5	
Total	**100**	