Contents

1. Introduction

In this assignment [1], an auction application based on smart contracts has been implemented in Solidity. Overall, three contracts have been written, namely: (1) *HashCalc* is used to calculate an SHA-256 hash to protect an auctioneer's bid commitment from eavesdropping; (2) *EthUsd* uses Provable API [2] (formerly called *Oraclize*) to request the ETH/USD exchange rate from an external website; (3) *Auction* brings the main mechanics of the auction.

The designed code has been deployed and tested on Remix website [3]. The archive with the code is attached to this report.

2. Description of the implemented functionality

2.1 Contract HashCalc

This contract contains only one function **function getHash(uint _bid, uint _seed, address _public_key) public pure returns (bytes32)**. It calculates the SHA-256 hash value of the input parameters.

2.2 Contract EthUsd

This contract is used to request the ETH/USD exchange rate from an external website. Two main functions of this contract are:

1. **function updatePrice() payable**: queries the exchange rate from `https://api.pro.coinbase.com/products/ETH-USD/ticker`.
2. **function getExchRate() public view returns(uint)**: returns the most recently requested exchange rate.

Other service functions and variables are commented in details in the attached files with the code. More information on the work of oracles may be found at [2].

2.3 Contract Auction

This contract implements the main functionality of the auction. It uses the interfaces to the above two contracts. In order to access them, it contains an object of the type *HashCalc* and an object of the type *EthUsd*. To initialize these objects, two functions, *function hashCalcExistAt(address _addr) public* and *function ethUsdExistAt(address _addr) public*, take the addresses of the deployed *contract HashCalc* and *contract EthUsd* respectively.

The *contract Auction* describes a structure *struct Auctioneer* that contains the complete information about a particular auctioneer, such as the status of registration (registered/not), the status of bid (committed/not), the hash of commitment, the value in Wei associated with the commitment, the value of bid in USD, the status of bid (sent/not), the status of the validity check for a bid (valid/not).

The *contract Auction* has the following private parameters:

1. **mapping(address => Auctioneer) auctioneers**: keeps track of auctioneers.

2. **address[] auctioneer_addresses**: a dynamic array of the addresses of auctioneers (is needed to 'parse' the stored data).

3. **HashCalc hashCalc** and *EthUsd ethUsdQuery* described above.

The *contract Auction* has the following public parameters:

1. **address public owner**: the owner of this contract.

2. **uint public ETHUSD**: exchange rate.

3. **uint public totalAuctioneers**: total number of auctioneer.

4. **bool public registrationClosed**: status of registration (open/closed).

5. **bool public commitmentClosed**: status of commitments (open/closed).

6. **address[] public winners**: a winner list.

7. **uint public totalBidsCommitted**: total number of committed bids.

8. **uint public totalBidsSent**: total number of sent bids.

The contract describes the following public functions:

1. **constructor () public**: assigns the owner of the contract.

2. **function registerAuctioneer(address _person) public**: registers an auctioneer.

3. **function computeHash(uint _bid, uint _seed) public view returns (bytes32)**: computes the hash of a commitment.

4. **function commitBid(bytes32 _hash_commit) public payable**: commit a bid (payable, because an auctioneer should add the value in Weis that is larger than his bid in USD).

5. **function sendBid(uint _bid, uint _seed) public**: send a bid.

The contract describes the following private methods with the modifier **modifier ownerOnly()** that prevents their launching by the auctioneers:

1. **function hashCalcExistAt(address _addr) ownerOnly public**: assigns the address of the contract that calculates the hashes of commitments.

2. **function ethUsdExistAt(address _addr) ownerOnly public**: assigns the address of the contract that queries the exchange rate.

3. **function registerStop() ownerOnly public**: stops registration.

4. **function commitBidStop() ownerOnly public**: stops commitment.

5. **function computeExchange() ownerOnly public payable**: requests the last exchange rate USD/Ethereum using an oracle.

6. **function getExchRate() public returns(uint)**: gets the last updated exchange rate.

7. **function computeWinner() ownerOnly public payable**: computes the winner.

8. **function resetAuction() ownerOnly public**: resets the auction by setting all the variables, except the winner list, to their initial values.

More detailed description of the above functions and variables is contained in the archive with the code.

3. Deployment

In this chapter, a test case for the developed code will be described.

Figure 3.1: Deploy all 3 contracts.

Figure 3.2: Initial values after deployment.

After all the contracts have been deployed, pass the addresses of two auxiliary contracts to *contract Auction*. Then, 3 auctioneers register themselves (the average price per 1 Eth is around $208):

1. Address = 0x22c0b77b2bD809f8e324c308f7B8c6D55f27700F.
   Valid bid (winner): bid = $700, seed = 5, value = 10 Eth (> bid).
2. Address = 0x089af33e51471E20bE9599f7a74e3224eadDa660.
   Invalid bid: bid = $800, seed = 6, value = 2 Eth (< bid).
3. Address = 0x8c2AA58fF1f9894d373e51C5D6df0753628d49C9.
   Valid bid (but less than the winning one): bid = $400, seed = 3, value = 5 Eth (> bid).

We expect the address 1 to be posted on the winner list at the end of the auction, the value sent from the address 2 to be hold as a penalty, and the address 3 to receive his/her value back.

Each auctioneer computes the hash for his/her bid, commits it together with the value. When all the hashed bids have been committed, auctioneers send their decrypted bids and seeds (see Fig. 3.3).

Figure 3.3: 3 users have committed and sent their bids.

After that, the auction authority closes the registration and the commitment. Then, the authority requests the exchange rate to check the bids sent by the users (see Fig. 3.4).

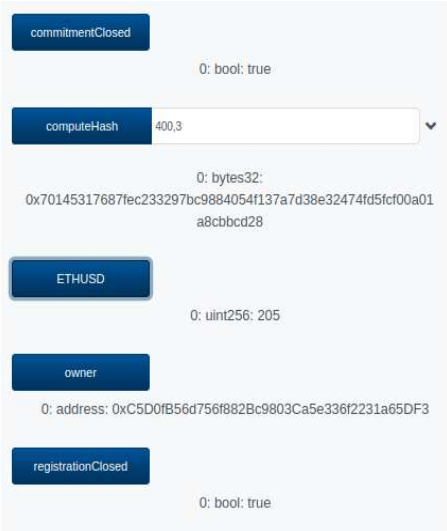

Figure 3.4: Registration and commitment are closed, exchange rate is updated.

The balances before the computation of the winner are shown on Fig. 3.5.

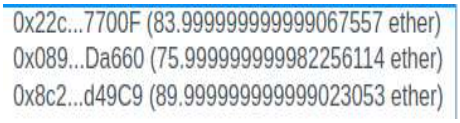

Figure 3.5: The balances for the address 1, the address 2 and the address 3 respectively before the validation of bids and the computation of the winner.

The balances after the computation of the winner are shown on Fig. 3.6.

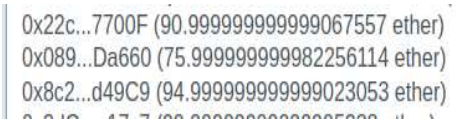

Figure 3.6: The balances for the address 1, the address 2 and the address 3 respectively after the validation of bids and the computation of the winner.

The winner list is presented on Fig. 3.7.

Figure 3.7: The winner list.

As we can see, the address 1 won as it was expected. The overhead was returned to the winner. The address 2 lost its money because of cheating. The address 3 got back its 5 Eth.

Finally, the authority resets all the variables, except the winner list.

Bibliography

[1] Y. Doroz, "Assigment 4: Smart Contracts," *WPI, ECE Department*, 2020.

[2] Provable, "Oracle Service API," 2020. Last accessed 27 April 2020.

[3] Remix, "Service for Solidity Smart Contracts' Deployment," 2020. Last accessed 05 May 2020.