

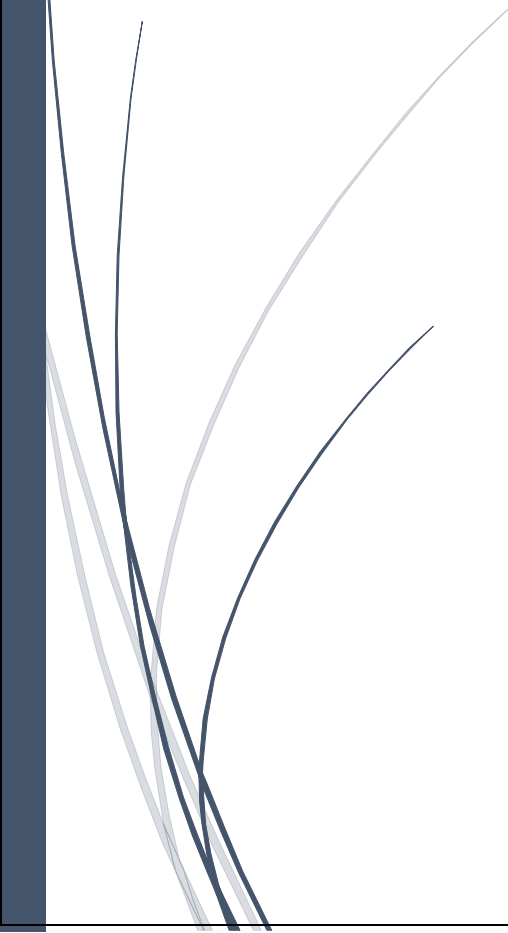


Assignment 3

Auction Smart Contract

Name : Monique Ehab

ID: 4928



Assumptions/ Handled Cases:

1. The auction manager determines the bidding time and the reveal time.
2. The auction manager deploys the auction.
3. The auction manager can't participate in the bid to prevent any personal influences.
4. The participants can only bid during the allowed bidding time.
5. Each participant can bid only once.
6. For a participant to take part in the auction he must pay a deposit value (value > predefined mindeposit) that is not returned until his commitment is revealed correctly as an **"incentive" to complete the auction**.
7. Each participant shares in the auctions by sending the deposit as the message value and a commitment, which is a hashed value of his public key (address), bidding value and nonce, in order to obtain the property of **"input independence"**.
8. Revealing only takes place once the bidding is over.
9. If the participant didn't reveal his commitment or didn't reveal it correctly, the deposit won't be refunded.
10. The auction can end only after the revealing time is over.
11. All the deposits of the participants who didn't reveal the commitments either correctly or at all are transferred to the auction manager.
12. Participants can't withdraw unless the auctionend function is called.
13. Each honest participant who didn't win can withdraw his own deposit.

Why didn't I use an automatic function to return the deposits to those who didn't win?

Based on search, this is what I found, it's a risky anti-pattern. The send or transfer methods can actually fail (for different reasons) and therefore, the payment loop won't end by successfully paying back all of the participants. For example, a malicious user that bid only 1 wei from a contract with a non-payable fallback can block the refund process by making the transfer method fail (and therefore throw) forever, meaning that the loop will never get executed completely.

14. In case the value of the deposit paid by the highest bidder is greater than the bidding value, in this case only he can withdraw the difference.
15. After a participant withdraws his deposit, the amount is set to 0, to prevent re-claiming the deposit again.

Code:

Part 1:

- Define some mappings and variables to be used later
- **The contract constructor :**
 - o Defines the auction manager as the one who deploys the auction (msg.sender).
 - o Takes the input of the bidding time and reveal time during deploy.

```
pragma solidity >0.4.23 <0.6.0;

contract Auction {

    mapping(bytes32 =>uint) blindBid;
    mapping(address =>bool) bidders;
    mapping(address => bytes32) public bids;
    mapping(address => uint) pendingReturns;
    mapping(address=>uint) honestbidders;
    uint mindep= 1;
    uint cheatersdeposit=0;
    uint public biddingEnd;
    uint public revealEnd;
    address[] honest;
    address[] allbids;
    address payable public auctionmanager;
    bool public ended;
    address public highestBidder;
    uint public highestBid;

    event AuctionEnded(address winner, uint highestBid);

    constructor(
        uint _biddingTime,
        uint _revealTime
    ) public {
        auctionmanager=msg.sender;
        biddingEnd = now + _biddingTime;
        revealEnd = biddingEnd + _revealTime;
    }
}
```

Part 2:

- **Function hash():**
 - o It computes the hash of the given concatenated values (address(public key)||value||nonce).
 - o It is a helping function because computing it using sha256 website and using it in the contract, results in different values during the reveal time due to padding issues.

- **Function bid():**
 - It requires:
 - Being called during bidding time and before reveal time.
 - Not being called by the auction manager.
 - Not being called more than once by the same participant.
 - Stores the address of the participants and the commitments.

```
function hash(address bidder, uint value, bytes32 nonce) public view returns(bytes32){
    bytes32 check= keccak256(abi.encodePacked(msg.sender,value,nonce));
    return check;
}

function bid(bytes32 _blindedBid)public payable {
    require(now < biddingEnd,"Bidding time ended.");
    require(msg.sender!=auctionmanager,"You are the auction manager. You can't bid.");
    require(!bidders[msg.sender],"You have already sent you bid");
    require(msg.value>mindep,"Message value is less than the minimum deposit!");

    blindBid[_blindedBid]=msg.value;
    bids[msg.sender]=_blindedBid;
    bidders[msg.sender]=true;
    allbids.push(msg.sender);
}
```

Part 3:

- **Function reveal():**
 - Requires the participants to provide their bidding value and the nonce value used.
 - If the computation of keccak256 is equal to the commitment of the participant, then he/she is considered an honest participant and can withdraw the deposit if he/she didn't win.

```
function reveal(uint value,bytes32 nonce) public payable {
    require(now > biddingEnd,"It is still bidding time.");
    require(now < revealEnd,"Revealing time ended.");

    bytes32 hashedBid = bids[msg.sender];
    bytes32 check= keccak256(abi.encodePacked(msg.sender,value,nonce));

    require(hashedBid == check,"NOT EQUAL. Deposit won't be refunded");

    honest.push(msg.sender);
    honestbidders[msg.sender]=value;
}
```

Part 4:

- **Function finalize():**
 - It is called after the reveal time is over, within the auctionEnd function.

- Uses the **checkBid** internal function to determine the highest bid and the address of the highest bidder.
- In case a participant is not the winner but is honest, his deposit and address are stored so he can withdraw them when the auction ends.
- If the participant is the winner but the deposit value is greater than the bidding value, the difference is stored so he can withdraw it after the auction ends.
- If the participant is a cheater, his deposit will not be available for withdrawal but will be saved aside to be added to the auction manager's account after the auction ends.

```
function finalize()internal{
    uint length= honest.length;
    for(uint i=0;i<length;i++){
        checkBid(honest[i],honestbidders[honest[i]]);
    }
    for(uint i=0;i<length;i++){
        if(honest[i]!=highestBidder){
            bytes32 hashedBid=bids[honest[i]];
            uint deposit=blindBid[hashedBid];
            pendingReturns[honest[i]]=deposit;
        }
        else if(honest[i]==highestBidder){
            bytes32 hashedBid=bids[honest[i]];
            uint deposit=blindBid[hashedBid];
            uint value=honestbidders[honest[i]];
            if(deposit>value){
                uint back=deposit-value;
                pendingReturns[honest[i]]=back;
            }
        }
    }
    uint length2=allbids.length;
    bool found;
    for(uint i =0;i<length2;i++){
        found=false;
        for(uint j=0;j<length;j++){
            if(allbids[i]==honest[j]){
                found=true;
            }
        }
        if(found==false){
            bytes32 hashedBid=bids[allbids[i]];
            cheatersdeposit+=blindBid[hashedBid];
        }
    }
}
```

```
function checkBid(address bidder, uint value) internal {
    if (value > highestBid) {
        highestBid = value;
        highestBidder = bidder;
    }
}
```

Part 5:

- **Function Withdraw():**
 - A participant can only withdraw after the reveal time is over and the auction ends.
 - An amount is withdrawn only if:
 - A participant is honest and not a winner.
 - A participant is a winner but (deposit > bidding value).
 - The amount is set to 0 afterwards to be withdrawn only once.
- **Function auctionEnd():**
 - Can be called only after the auction ends.
 - Finalize() function is implemented.
 - The auction manager gets the highest bid and the deposits of cheater participants.

```
function withdraw() public {
    require(now > revealEnd, "Revealing time didn't end yet.");
    require(ended, "Auction didn't end yet.");
    uint amount = pendingReturns[msg.sender];

    pendingReturns[msg.sender] = 0;

    msg.sender.transfer(amount);
}

function auctionEnd() public {
    require(now > revealEnd, "Revealing time didn't end yet.");
    require(!ended);
    finalize();
    emit AuctionEnded(highestBidder, highestBid);
    ended = true;
    auctionmanager.transfer(highestBid);
    auctionmanager.transfer(cheatersdeposit);
}
```

Bonus Part:

- **Assumption :**

- The record provided by the auction manager is valid.

- **Method:**

- We can use the digital signature, the auction manager protects the record, from the access of any of the participants, so it can't be accessed except by sending the digital signature of the owner.
- And for the auction Manager to obtain the highest bid, he will need also to provide his digital signature.
- So this signature will be used at the end of the auction for two reasons:
 - Transfer the highest bid to the auction manager.
 - Grant access of the record to the winner.
- And this method will be time bound, in case the auction manager didn't provide his signature within the time limit, the participant can withdraw his bid.

- **Security aspects handled:**

- Privacy of the record from access or manipulation of attackers or cheating participants.
- Maintaining the incentive that will urge the auction manager to continue the protocol to obtain the bid and the right of the participant in obtaining the record, simultaneously.
- Guarantee that the participant won't lose his money in case the auction manager didn't continue the auction.