**048888 – Winter 18/19 – Ameer and Jiaqi**

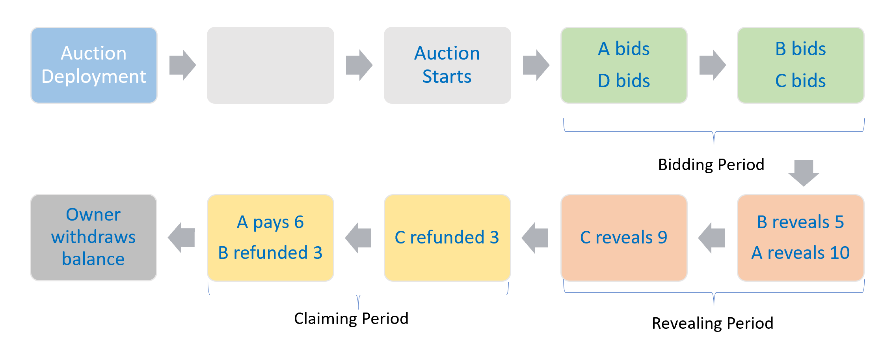
**Second price sealed bid auction smart contract:**

**Overview:**

A second price sealed-bid auction is a blind auction in which all bidders simultaneously submit sealed bids, so that no bidder knows the bid of any other participant. The highest bidder pays the second highest bid, bidding truthfully is the dominant strategy of this auction.

Auction smart contract is a great use case as it provides transparency and is trustless. In brief, the contract can be deployed by any party offering to sell an asset and will be open for bids for a limited time frame. The bids are hashes of the sealed offers, every bidder needs to provide a fixed deposit to be escrowed in case he fails to complete the auction. Once the bidding period ends, the bidders have a time frame to provide their real bids, once the offers are verified, a winner is selected and has a fixed time frame to pay for the asset or the auction is cancelled and he loses the deposit. All losing bidders will be refunded. We assume that the party that deployed the smart contract is trusted using the smart contract for better transparency.

**Design considerations:**

* Commitment scheme: To seal the bids while the state of the system is public we use double hash of the bid and salt: Hash(Hash(bid) || Hash(salt)).
* Escrow: The escrow fund is determined by the seller and it’s lost if the bidder fails to reveal a valid bid, it also provides spoof protection for some degree.
* Time: block timestamps are set by miners and are not necessarily safe from spoofing. A best practice is to demarcate time based on block number. We know with a fair amount of certainty that Ethereum blocks are generated roughly every 15 seconds.
* Gas: to reduce storage gas, we only store an IPFS file hash. The data referred to by this hash may contain things like the auction’s title and description, photo URLs, etc.
* Reserved price: The contract sets a minimum price for the auctioned item.
* Tie breaking rule: first one to reveal wins.
* Refund policy: We know from the DAO hack that it’s always best to stick to a “withdrawal” pattern, this helps us to avoid re-entrance bugs that could cause unexpected behavior.

**Auction steps:**

**Potential improvements:**

* Add buyer protection: Seller delivers to get deposit refund and buyer confirms receipt to get escrow refund. This method is asymmetric and gives the buyer an edge, thus need to be further studied.
* Implement ERC20 Auction Token to be used in the auctions instead of ether.

Tools used: Remix, Ganache, myEtherWallet, Truffle, Metamask, Ropsten test net.

Code is in appendix.

**Miscellaneous**

**How we split the work?**

Most of the time we sat together to work on the project. We did research separately on how to implement such a smart contract and made a discussion about the design and agreed on the main structure of the project. Ameer started writing the code and Jiaqi helped him find solutions whenever there were technical difficulties. In the end we tested the code, debugged and did simulation together.

**Challenges**

The challenges are mainly threefold, the design of the smart contract, the implementation (coding), and testing. Regarding the design, we had to figure out how to make certain mechanism that are easy to implement in real life work in a smart contract (e.g. a sealed-bid could be put in an envelope in real life but appear as a hashed number in a smart contract). As for coding, since Solidity is a relatively new language, we found the explanation for certain syntax rather unclear. For example, the usage of hash function keccak256() was not specified and no example was given in the official documents. The environment and tools used to test the code also took us quite some time to get used to, we checked the Ethereum java script libraries and spent relatively long time figuring out how to deploy and interact with the contract on ganache and other test nets.

**What I personally learned from the project**

**Jiaqi**

The concept of smart contract was new to me and I find it fascinating. Before I did this project, I didn’t completely understand how the blockchain works and what things we could do with it. Besides, this project reinforced my understanding of some key concepts in cryptocurrency such as the usage of hashing.

**Ameer**

I had some background with smart contracts and blockchain in general but never had the chance to implement and deploy a smart contract, so It was a great chance to work on development and see a working system. It also helped me understand things that were not clear to me such as the difference between bitcoin addressing and Ethereum addresses and the EVM in general, and see some of the limitations of this Turing complete machine.

Appendix: Code

pragma solidity >=0.4.21 <0.6.0;

contract sealedAuction {

//static auction info

address payable public owner;

uint256 public escrowAmount;

uint256 public ipfsHash;

bool public started;

uint256 public startBlock;

//timeline

uint256 public biddingDuration;

uint256 public revealingDuration;

uint256 public claimingDuration;

address public winner;

bool public winnerSettled;

uint256 public reservedPrice;

uint256 public maxRevealedBid;

uint256 public secondRevealedBid;

mapping(address => bytes32) public sealedBids;

mapping(address => uint256) public escrowedFunds;

mapping(address => uint256) public revealedBids;

//function() payable { } //fallback function

constructor(uint256 \_ipfsHash, uint256 \_escrowAmount, uint256 \_biddingDuration, uint256 \_revealingDuration,

uint256 \_claimingDuration, uint \_reservedPrice) public

{

require(\_biddingDuration > 0);

require(\_revealingDuration > 0);

require(\_claimingDuration > 0);

owner = msg.sender;

ipfsHash = \_ipfsHash;

escrowAmount = \_escrowAmount;

biddingDuration = \_biddingDuration;

revealingDuration = \_revealingDuration;

claimingDuration = \_claimingDuration;

reservedPrice = \_reservedPrice;

started = false;

maxRevealedBid = 0;

secondRevealedBid = 0;

winnerSettled = false;

}

function startAuction() public

isOwner

notStarted

{

startBlock = block.number;

started = true;

//emit auctionStarted(startBlock, biddingDuration);

}

//event auctionStarted(uint256 startBlock, uint256 biddingDuration);

function placeBid(bytes32 bid) public

payable

isNotOwner

inBiddingPeriod

sufficientEscrow

{

sealedBids[msg.sender] = bid;

escrowedFunds[msg.sender] += msg.value;

}

function revealBid(uint256 bid, uint256 salt) public

inRevealingPeriod

{

require(bid >= reservedPrice);

require(keccak256(abi.encodePacked(keccak256(abi.encodePacked(bid)),keccak256(abi.encodePacked(salt)))) == sealedBids[msg.sender]);

revealedBids[msg.sender] = bid;

if(bid > maxRevealedBid){

secondRevealedBid = maxRevealedBid;

maxRevealedBid = bid;

winner = msg.sender;

}

else if(bid > secondRevealedBid){

secondRevealedBid = bid;

}

}

function settle() public

payable

inClaimingPeriod

isWinner

winnerNotSettled

{

if(escrowedFunds[msg.sender] >= secondRevealedBid){

require(msg.sender.send(escrowedFunds[msg.sender] - secondRevealedBid));

}

if(escrowedFunds[msg.sender] < secondRevealedBid){

require(msg.value >= (secondRevealedBid - escrowedFunds[msg.sender]));

}

winnerSettled = true;

}

function refund() public

inClaimingPeriod

isNotWinner

hasValidBid

hasEscrowedFunds

{

require(msg.sender.send(escrowedFunds[msg.sender]));

escrowedFunds[msg.sender] = 0;

}

function closeAuction() public

isOwner

afterclaimingPeriod

{

msg.sender.transfer(address(this).balance);

}

modifier isOwner {

require(msg.sender == owner);

\_;

}

modifier isNotOwner {

require(msg.sender != owner);

\_;

}

modifier notStarted {

require(!started);

\_;

}

modifier inBiddingPeriod {

require(started && block.number > startBlock && block.number <= startBlock + biddingDuration);

\_;

}

modifier inRevealingPeriod {

require(started && block.number > startBlock + biddingDuration && block.number <= startBlock +

biddingDuration + revealingDuration);

\_;

}

modifier inClaimingPeriod {

require(started && block.number > startBlock + biddingDuration + revealingDuration && block.number <= startBlock +

biddingDuration + revealingDuration + claimingDuration);

\_;

}

modifier afterclaimingPeriod {

require(started && block.number > startBlock +

biddingDuration + revealingDuration + claimingDuration);

\_;

}

modifier sufficientEscrow {

require(msg.value >= escrowAmount);

\_;

}

modifier isWinner {

require(msg.sender == winner);

\_;

}

modifier isNotWinner {

require(msg.sender != winner);

\_;

}

modifier hasValidBid {

require(revealedBids[msg.sender] != 0);

\_;

}

modifier hasEscrowedFunds {

require(escrowedFunds[msg.sender] > 0);

\_;

}

modifier winnerNotSettled {

require(!winnerSettled);

\_;

}