BLIND AUCTION

- Instead of an actual bid, a bidder sends its hashed version.
- Hence we can't track back the address from which a bidder participate in the bidding process and can conduct multiple bidding.
- There is also no time pressure as the end time of the auction approaches.

AIM

- Develop a smart contract for blind auction

ALGORITHM

- Define struct for Bid
- For the bidding process, make address of beneficiary, ending time of bidding and revelation of bidding be public
- Write constructor with respect to the struct defined
- Make it's possible to place multiple bids from a single address by using keccak256 encryption
- Write functions for
    - BID, REVEAL BID, PLACEBID, WITHDRW FUND, REFUND TO GIVE FOR TOP BIDDERS WHO CANNOT WIN THE BIDDING, DISPLAY HIGHEST BIDDER DETAILS & ENDAUCTION

```
pragma solidity >0.4.23 <0.7.0;

contract BlindAuction {
    struct Bid {
        bytes32 blindedBid;
        uint deposit;
    }

    address payable public beneficiary;
    uint public biddingEnd;
    uint public revealEnd;
    bool public ended;

    mapping(address => Bid[]) public bids;

    address public highestBidder;
    uint public highestBid;
```

allow the participants to withdraw the bids that didn’t win:

```
    mapping(address => uint) pendingReturns;

    event AuctionEnded(address winner, uint highestBid);
```

It is recommended to validate function inputs. You can easily do that by using function modifiers. We apply `onlyBefore()` to the `bid()` below.

underscore in the modifier’s body, turning it into a new function body:

```
    modifier onlyBefore(uint _time) { require(now < _time); _; }
    modifier onlyAfter(uint _time) { require(now > _time); _; }
```

include a `constructor` to define auction details:

```
 constructor(
        uint _biddingTime,
        uint _revealTime,
        address payable _beneficiary
```

```
    ) public {
        beneficiary = _beneficiary;
        biddingEnd = now + _biddingTime;
        revealEnd = biddingEnd + _revealTime;
    }
```

To place a blinded bid, you need `` `_blindedBid` = keccak256(abi.encodePacked(value, fake, secret)) ``.

It’s possible to place multiple bids from a single address.

If Ether that you send along with the bid is `value` and `fake` is set to `false`, the bid is considered valid.

To hide your real bid but make a deposit, you can either set `fake` to `true`, or send a non-exact amount.

```
function bid(bytes32 _blindedBid)
        public
        payable
        onlyBefore(biddingEnd)
    {
        bids[msg.sender].push(Bid({
            blindedBid: _blindedBid,
            deposit: msg.value
        }));
    }
```

you will only get a refund if your bid can be revealed correctly after the auction.

use `reveal()` to see the blinded bids.

Refunds will be available for all topped bids, as well as invalid bids that were blinded properly:

```
function reveal(
        uint[] memory _values,
        bool[] memory _fake,
        bytes32[] memory _secret
    )
        public
        onlyAfter(biddingEnd)
        onlyBefore(revealEnd)
    {
        uint length = bids[msg.sender].length;
        require(_values.length == length);
        require(_fake.length == length);
        require(_secret.length == length);

        uint refund;
        for (uint i = 0; i < length; i++) {
            Bid storage bidToCheck = bids[msg.sender][i];
            (uint value, bool fake, bytes32 secret) =
                    (_values[i], _fake[i], _secret[i]);
```

If the bid cannot be revealed, there will be no refund as well.

also make sure it’s impossible to claim the same refund more than once:

```
if (bidToCheck.blindedBid != keccak256(abi.encodePacked(value, fake, secret))) {
                continue;
            }
            refund += bidToCheck.deposit;
            if (!fake && bidToCheck.deposit >= value) {
                if (placeBid(msg.sender, value))
                    refund -= value;
            }
            bidToCheck.blindedBid = bytes32(0);
        }
        msg.sender.transfer(refund);
    }
```

The `placeBid()` function is internal: that means you can only call it from the contract or others derived from it. how to make sure to refund the outbid offer:

```
function placeBid(address bidder, uint value) internal
            returns (bool success)
    {
        if (value <= highestBid) {
            return false;
        }
        if (highestBidder != address(0)) {
            pendingReturns[highestBidder] += highestBid;
        }
        highestBid = value;
        highestBidder = bidder;
        return true;
    }
```

include `withdraw()` for withdrawing a topped bid and set **amount > 0** :

```
function withdraw() public {
        uint amount = pendingReturns[msg.sender];
        if (amount > 0) {

            pendingReturns[msg.sender] = 0;

            msg.sender.transfer(amount);
        }
    }
```

Finally,finish our auction and send the highest bid to the beneficiary:

```
function auctionEnd()
        public
        onlyAfter(revealEnd)
    {
        require(!ended);
        emit AuctionEnded(highestBidder, highestBid);
        ended = true;
        beneficiary.transfer(highestBid);
    }
}
```

SMART CONTRACT

```
pragma solidity >0.4.23 <0.7.0;

contract BlindAuction {
    struct Bid {
        bytes32 blindedBid;
        uint deposit;
    }

    address payable public beneficiary;
    uint public biddingEnd;
    uint public revealEnd;
    bool public ended;

    mapping(address => Bid[]) public bids;

    address public highestBidder;
    uint public highestBid;

    mapping(address => uint) pendingReturns;

    event AuctionEnded(address winner, uint highestBid);

    modifier onlyBefore(uint _time) { require(now < _time); _; }
    modifier onlyAfter(uint _time) { require(now > _time); _; }

 constructor(
        uint _biddingTime,
        uint _revealTime,
        address payable _beneficiary
    ) public {
        beneficiary = _beneficiary;
        biddingEnd = now + _biddingTime;
        revealEnd = biddingEnd + _revealTime;
    }
```

```
function bid(bytes32 _blindedBid)
        public
        payable
        onlyBefore(biddingEnd)
    {
        bids[msg.sender].push(Bid({
            blindedBid: _blindedBid,
            deposit: msg.value
        }));
    }

function reveal(
        uint[] memory _values,
        bool[] memory _fake,
        bytes32[] memory _secret
    )
        public
        onlyAfter(biddingEnd)
        onlyBefore(revealEnd)
    {
        uint length = bids[msg.sender].length;
        require(_values.length == length);
        require(_fake.length == length);
        require(_secret.length == length);

        uint refund;
        for (uint i = 0; i < length; i++) {
            Bid storage bidToCheck = bids[msg.sender][i];
            (uint value, bool fake, bytes32 secret) =
                    (_values[i], _fake[i], _secret[i]);

if (bidToCheck.blindedBid != keccak256(abi.encodePacked(value, fake, secret)))
{
                continue;
            }
            refund += bidToCheck.deposit;
            if (!fake && bidToCheck.deposit >= value) {
                if (placeBid(msg.sender, value))
                    refund -= value;
            }
            bidToCheck.blindedBid = bytes32(0);
        }
        msg.sender.transfer(refund);
    }

function placeBid(address bidder, uint value) internal
            returns (bool success)
    {
        if (value <= highestBid) {
            return false;
        }
        if (highestBidder != address(0)) {
            pendingReturns[highestBidder] += highestBid;
        }
        highestBid = value;
        highestBidder = bidder;
        return true;
    }


function withdraw() public {
        uint amount = pendingReturns[msg.sender];
        if (amount > 0) {

            pendingReturns[msg.sender] = 0;

            msg.sender.transfer(amount);
        }
    }

function auctionEnd()
        public
        onlyAfter(revealEnd)
    {
        require(!ended);
        emit AuctionEnded(highestBidder, highestBid);
        ended = true;
        beneficiary.transfer(highestBid);
    }
}
```