Blockchain-based, Privacy-preserving, First Price Sealed Bid Auction (FPSBA) Verifiable by Participants

by
Ehsan Ghasaei

A Report Submitted in Partial Fulfillment of the Requirement for the Degree of
Master of Engineering (MEng)
In the Department of Electrical and Computer Engineering

University of Victoria

Supervisory Committee

Dr. Amirali Baniasadi,
Supervisor
(Department of Electrical and Computer Engineering)

Dr. Daler N. Rakhmatov,
Departmental Member
(Department of Electrical and Computer Engineering)

Abstract

Auction is becoming a prevalent method for trading goods. Bidding-based auctions can assist sellers and buyers in identifying the real value of the asset. Utilizing the internet, auctions have been modernized and become e-auctions. Although e-auction is becoming a popular platform for online bidding, the platform is prone to multiple threats and is potentially vulnerable. For example, data generated from e-auction is being stored in a central database, which is often at the risk of being compromised or failure. In addition, the central data center might act maliciously and leak sensitive information to one of the bidders.

Utilizing the blockchain concept in the e-auction systems can mitigate the aforementioned threats. Blockchain-based auction systems will be up and running due to blockchain's distributed nature, i.e., such e-auction systems are more resilient. Moreover, thanks to the hash functions, embedded in blockchain, and the immutability of the blockchain, there is a negligible chance for cheating. While blockchain brings about previously-mentioned value proposition for e-auctions, the method can have shortcomings as well. First and foremost, lack of privacy is a well-known challenge for blockchain-based e-auctions owing to the transparent nature of the blockchain network. Bids submitted by users are visible to the world and this can violate user's privacy.

To address this controversial issue of lack of privacy, in this work, an improved version of blockchain-based e-auction is designed in a fashion that users submit concealed value of their bids into the blockchain and will send the bids in plaintext format to an off-chain code. The off-chain code will calculate the winner. The winner is then verified by each participant.

Compared to other blockchain-based auctions, the proposed methodology helps preserving user privacy. Also, integrity of the off-chain is assessed by each bidder and if participants identify any flaw in the performance of the off-chain code, the participants can dispute.

Table of Content

List of Figures

List of Tables

Acknowledgments

I would like to thank my supervisor, Dr. Amirali Baniasadi, for his endless support, valuable insight and suggestions, and constant guidance throughout my Meng studies.

I also would like to thank my precious parents and three brothers for their high-value motivations and supports not only in my studies, but in my entire life.

Chapter 1: Introduction

Auction is among the prevalent methods for trading goods. The bidding process can contribute to the determination of the real value of the product. Running auction systems on the internet can bring about multiple advantages and overcome limitations associated with conventional auction systems. For instance, people located at different geographical locations can participate in an auction easily and efficiently [1]. In addition, electronic auctions (e-auctions) take place without requiring any physical area. Hence, utilization of e-auction can be cost efficient [1]. E-auctions are mainly classified as: open e-auctions and sealed-bid e-auctions [1]. In open bidding process, bidders are aware of other bids' values and act accordingly. The auction resumes and the bidding price increases until no one is willing to pay. Then, the bidder with the highest suggesting price is the winner. In the sealed-bid e-auction, the bids are submitted in sealed envelopes. Thereby, no one except the auctioneer or the platform has access to the submitted values. After submission of the bids, the envelopes will be opened and the bidder with the highest suggesting price will be the winner.

Both open-bidding and sealed-bidding e-auctions bring about a number of advantages; however, both are subject to two major drawbacks [2]. The first drawback is that e-auctions need a trusted third party to store electronic data and to execute auction websites and platforms. This can increase cost, risk of data breach, and risk of data loss. The second drawback is that all participants must trust the bidding procedure while there is always a chance of corruption [2]. For example, instead of revealing the real values, the trusted entity may reveal fake data. This behavior can lead to malicious activity conducted by the participants.

Blockchain is a distributed peer to peer network [2]. The distribution feature of blockchain can help digital solutions to eliminate the risk of data loss or in a more extreme case, failure of the entire data server; since each peer in blockchain can operate as an independent data server. Moreover, every data stored on the blockchain is visible and immutable which significantly reduces the chance of malicious activities and enhances the transparency of the e-auction [2].

1.1 Problem Definition and Motivation

While blockchain can address the drawbacks associated with current e-auctions, the blockchain escalates the possibility of bidders' privacy violation [2,3]. Information gathered by blockchain is publicly visible and accessible. For blockchain-based auctions, this means that the users' bids are publicly visible. Malicious actors such as hackers can obtain this sensitive information to gain insight about the user's monetary asset and have a clear understanding about users' financial activities over the blockchain network. In addition, the bidders can study the losing bids from the previous auctions to have a better prediction on how much value each bidder owns and use this information against them in the next auctions, this learning curve can put the concept of a fair bidding in jeopardy [4].

In this research, the proposed approach will help preserving users' privacy over a blockchain-based e-auction system. Instead of submitting bids in plaintext, the participants submit masked bids into the blockchain. Hence, the blockchain observers will have no knowledge of the bid values. In addition, an off-chain code is developed, evaluated, and executed to find the highest bid. The highest bid is validated by each and every participant. All the interactions occurring in this auction system are saved into blockchain.

The rest of the report is structured as follows: Chapter 2 describes different classes of e-auctions and provides a basic concept of the blockchain. In chapter 3, the proposed architecture, implementation methodology and tools such as Truffle, Infura, Rinkeby test net will be comprehensively explained. Chapter 4 evaluates the cost and efficiency of the proposed approach. Finally in chapter 5, the conclusion and future work are discussed.

Chapter 2: Auction and Blockchain Review

2.1 Auction Review

Auction can be divided into four groups [5]:

1. English auction (open auction)
   This is an open cry auction [1][5]. The auctioneer begins the competition at the lowest accepting price. Then, the bidders raise their hands to show interest on a higher price. This is continued until no one raises their hands. At that moment, the bidder who was last to raise their hand and proposed the highest price is the winner and must pay the corresponding price [5].
2. Dutch auction (open auction)
   This is an open cry auction [1][5]. The auctioneer commences at a reasonably high price. Then the auctioneer starts decreasing the price and interested bidders raise hands. The first participant who has their hand raised is the winner and must pay the associated price [5].
3. First price sealed bid auction (FPSBA) (sealed bid auction)
   This is a sealed bid auction [1][5]. In this type of auction, bidders put their bids inside an envelope. Then, the auctioneer collects all the envelopes and reveals them. The bidder with the highest offering is the winner and must pay the price. This report will study this type of auction.
4. Vickery auction (sealed bid auction)
   This is a sealed bid auction [1][5]. This auction has the same steps as the previous auction (FPSBA). However, the winner will pay the second highest offer [6].

2.2 Blockchain Review

Blockchain is a peer-to-peer distributed ledger publicly available. All transactions and data intended to be stored on blockchain must undergo a particular process called mining. After passing the process, a bulk of data is added to a block. Each new block connects to the previous blocks. Any change to a confirmed data in a block result in alteration and inconsistency in the previous blocks and is ignored by the active nodes. Therefore, once a data is submitted to the blockchain, it cannot be deleted or tampered with [7].

2.2.1 Ethereum

The two mostly used blockchain platforms are Ethereum and Bitcoin [7]. Ethereum can be used for cryptocurrency exchange and is also a programmable framework that businesses in financial, art, gaming, technology etc., can utilize to integrate into blockchain [7]. Source codes developed in Ethereum are compiled into EVM (Ethereum Virtual Machine) bytecodes [7]. Eth is the native currency of Ethereum [8]. Executing codes on blockchain requires paying a fee called Gas. Gas is calculated in Eth and is convertible into USD.

2.2.2 Transactions

Transactions are activities on blockchain networks initiated by humans. The transactions change the EVM (Ethereum Virtual Machine) global state. For example, Bod sends 1 eth to Alice, and this is called a transaction resulting in changing the EVM world state [7][9].

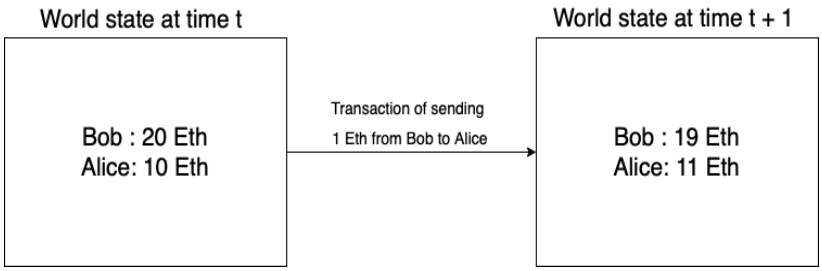

Figure 1. Sample of transaction changing the EVM world state

Each transaction has a fee that must be paid by the transaction initiator. The fee might be as simple as sending Eth or as complicated as performing sophisticated mathematical expressions. The transactions contain the following information [7][9]:

- Recipient: the receiver address. There are two types of recipients: Externally owned account and contract address. Externally owned accounts refer to addresses supervised by humans. Transactions with externally owned accounts recipient can only send and receive Eth. Transactions having a contract address recipient executes the contract code.
- Signature: signature confirms if the authorized sender has initiated the transaction.
- Value: The value specifies the amount to be sent or received if the transaction recipient is an externally owned account.
- Data: any input data for the transactions with the contract address recipient. This field is optional.
- Gas limit: gas limit is the computational threshold associated with each transaction.

The transactions process comes as follows [9]:

1. The user initiates the transaction and sends it to one of the blockchain nodes.
2. The node adds the transaction into Tx-pool (all pending transactions are collected in this pool) and broadcasts it to all active nodes.
3. All nodes receive the transaction and place it into their Tx-pool.
4. The nodes start a competition: they validate the transactions and then start solving a cryptographic puzzle (mining). The winner submits the transactions into the blockchain and broadcasts a copy of the new blockchain.
5. The nodes receive the copy, confirm, and add the transaction into the blockchain.

2.2.3 Hash Function

Hash functions are mathematical functions mapping a free sized value into a fixed size output called message digest (also known as hash value) as shown in figure 2 [10].

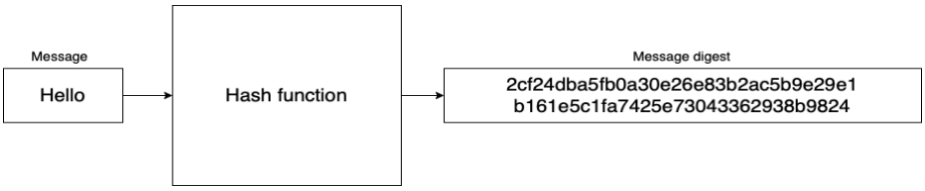

Figure 2. The hash value of "Hello" word

Hash functions have the following properties:

1. Fixed length output [10]
   Any hash function takes the input in any size and converts it to a hash value with the fixed size. The size can vary from 160 to 512 bits.
2. Pre-Image resistance [10]
   Having a hash function *h* and output *y*, it is computationally difficult to find *x* such that *h(x)=y*. In other words, having a message digest, it is hard to move reversely and find the corresponding message.
3. Second pre-image resistance (collision resistance) [10]
   Given a hash function *h* and input *x* and output *y*, it is difficult to find *z* such that h(z) = h(x) = y.
4. Non-correlation of input and output
   Given a hash function *h* and input *x* and output *y* (*h(x) = y*), a slight change to x results in a totally different output z.

2.2.4 Blockchain Architecture

As shown in figure 3, each block has the following parameters [2][7]:

1. Time stamp: The time this block is created.
2. Previous block hash: The hash of the previous block
3. Current block hash: The hash of the current block
4. Nonce: Nonce is the value that a node finds in solving the cryptographic puzzle (mining) and adds it to the block for further verifications.

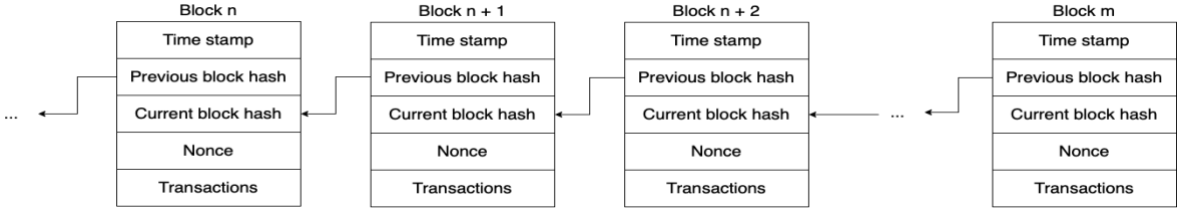

Figure 3. Blockchain architecture

Any change to a data by a malicious actor causes a change in the hash of the associated block. Then, the data in the next block is altered since the blocks are chained using the hash. This effect is continued in all the following blocks. In other words, any small change into an existing block causes it to have a completely different blockchain compared to other nodes. Any block added to the altered blockchain by the malicious party will be rejected by other active nodes due to unequal copy of blockchains. Therefore, blockchain is resistant against alteration or deletion.

2.2.5 Peer-to-Peer Network (P2P)

Peer to peer network is a distributed communication model [11] which is the core foundation of blockchain. In the centralized model, the actors can either have client or server roles. Client requests the data and server processes it while also storing the data. However, P2P network consists of nodes that can act as a client or server [11].

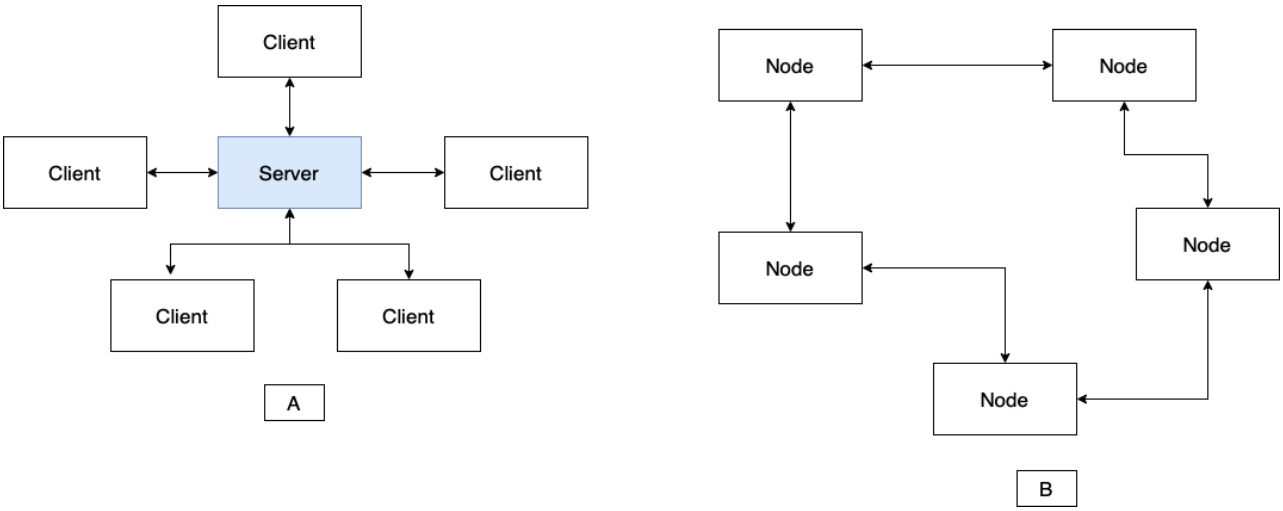

Figure 4. A) Client server network B) Peer-to-Peer network

Figure 4 shows the client/server (A) and P2P architecture (B). In client/server, the failure of the server ends with the failure of the whole system. However, in P2P, all nodes can act as servers and recover node failure. This is how blockchain is resistant against data loss and system failure.

2.2.6 Mining (Proof of Work)

Each node is located in different cities or countries and stores a local copy of the global blockchain. The main challenge is that due to this geographical difference, they might not receive the transactions in the same order. This will result in nodes having a different block and consequently, a totally different blockchain. Hence, it is critical for the nodes to reach consensus [12]. To do so, the nodes participate in mining. Mining is the work of solving a hashed puzzle. The miners utilize the non-correlation feature of the hash functions and adjust the nonce in the puzzle to find the answer [12]. Proof of work (POW) is the most used mining technique in Ethereum [12]. The benefits and drawbacks of proof of work come in table 1 [12].

Table 1. Advantages and disadvantages of POW

Advantages	Disadvantages
Controlling the flow of adding blocks by adjusting the puzzle difficulty	POW requires a particular and expensive system configuration
Provides a fair and random method for adding the blocks	Proof of work consumes a huge amount of energy. For example, Ethereum consumes 73.2 TWh annually, equal to the energy required by middle sized country such as Austria

Note: It is noteworthy to mention that proof of stake (POS) will replace proof of work in 2023 [12].

2.2.7 Ethereum Virtual Machine (EVM)

Ethereum virtual machine (EVM) is a virtual environment running on each node's system and is responsible for executing and deploying smart contracts. At the time of receiving smart contract calls, EVM executes the smart contract codes [13].

2.2.8 Accounts, Addresses, and Wallets

Ethereum accounts are divided into two groups [14]:

1. Externally owned accounts (EOA): Users control this account. Each EOA consists of a pair of public and private key.
2. Smart contract accounts: These accounts belong to the smart contract and are controlled by code. Unlike EOA, smart contract accounts do not have any associated private key. Externally owned accounts can interact with smart contract accounts by calling the methods.

An address in Ethereum is generated using the following steps [14]:

1. Hash256 of the public key is calculated.

$$y = hash256(publickey)$$

2. Last 20 bytes of the hash output are selected as shown in figure 5.

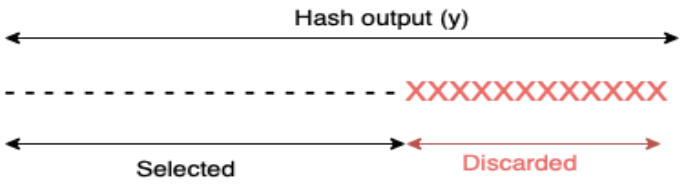

Figure 5. How hash output is changed in the process of generating Ethereum address

3. 0x is added to the beginning of the output in step 2.

Ethereum wallets are interfaces for accessing the accounts and initiating transactions. Wallets can contain multiple accounts.

2.2.9 Smart Contract

The main feature distinguishing Ethereum from other blockchain networks is smart contracts. Smart contracts are computer programs located in Ethereum forcing the users to obey certain rules. [15].
Smart contract consists of two parts [15]:

1. Data: information related to the contract.
2. Functions: functions or methods are responsible for performing actions and changing the global state (data) of blockchain.

Smart contracts are Ethereum accounts meaning that they can have balance, send, or receive transactions [15].

Companies can utilize smart contracts to integrate their business logic into the blockchain world. In other words, smart contracts connect businesses to the blockchain.

2.2.10 Solidity

Solidity is a programming language for developing smart contracts. There are other programming languages apart from solidity for such purpose, such as Vyper and Yul [16].

Solidity has the following features [16]:

1. Object oriented: smart contract instances act as objects.
2. High-level language impacted by C++ and similar to Python, JavaScript, and C++.
3. Supports inheritance, library extension and sophisticated user types.
4. Statically typed: developers must define the type of the variable.

2.2.11 Remix

Remix is an open-source IDE (integrated development environment) for developing smart contracts in Solidity. In addition, it provides a test environment containing Ethereum accounts for assessing the written code. Remix comes with great plugins for connecting third party apps such as Git [17][18].

2.2.12 Metamask

Metamask is a popular wallet for storing multiple Ethereum accounts.

Metamask comes with the following features [18][19]:

1. Accessing Ethereum accounts.
2. Sending and receiving transactions.
3. Blockchain users allow Metamask to connect to blockchain-based web applications and perform transactions easily and securely. This will relieve the users of transaction signatures, technical difficulties etc.

2.2.13 Truffle

Truffle makes the blockchain development more straightforward as it provides the following features [20][21]:

1. Local blockchain environment to test the smart contract without paying any unnecessary gas fee with the help of complementary tools named Ganache, and Drizzle [20].
2. Developing test script: programmers can write tests using Mochajs [22]. This greatly helps them as they utilize a well-known testing framework (Mochajs) to evaluate the smart contract. Hence, Truffle will relieve the developers of the process of learning tests in Solidity.
3. Once the smart contract meets the expectations, the developers can use Truffle to deploy it into worldwide blockchain networks.

2.2.14 Infura

Infura helps developers and engineers to smoothly deploy the applications into blockchain. What Infura eliminates is the need for developers to configure blockchain nodes resulting in high cost and space. In combination with IPFS, Infura successfully handles large files scaling on blockchain [23, 24].

2.2.15 Rinkeby

Blockchain networks are divided into two groups:

1. Test network (test net): this network is designed for developers and engineers to test their decentralized application in a test blockchain network. Some examples of the test net are Rinkeby [25], Ropsten, Kovan etc.
   Note: This project leverages Rinkeby network.

2. Main network (main net) [26]: When the decentralized application fulfills the requirements in test net, it can be deployed into main net. In other words, the product enters the production phase.

Chapter 3: Proposed Methodology

3.1 System Architecture

Submitting bids in plaintext into blockchain is where the privacy issue arises.
The main idea of the proposed methodology is that instead of bids, the participants submit the committed hash of the bids. This will ensure no malicious activity by the bidders. In addition, the bidders send the bids to off-chain code. The off-chain code finds the highest bid and returns it back to the participants. Upon receiving the highest value of the bids, each client compares it with their own bid as explained in table 2.

Note: this comparison is executed by every single bidder.

Table 2. Comparing the highest value of the bids and user's suggested bid

Comparing result	Explanation
Highest value > user's value	It means the bidder has validated the highest value. This bidder waits until the winner is announced.
Highest value == user's value	It means the bidder is the winner. This bidder submits the winner information for further verifications.
Highest value < user's value	It means the bidder has detected a flaw in the off-chain and acts as disputer. This disputer reveals the information for claim verification.

The basic system architecture comes in figure 6.

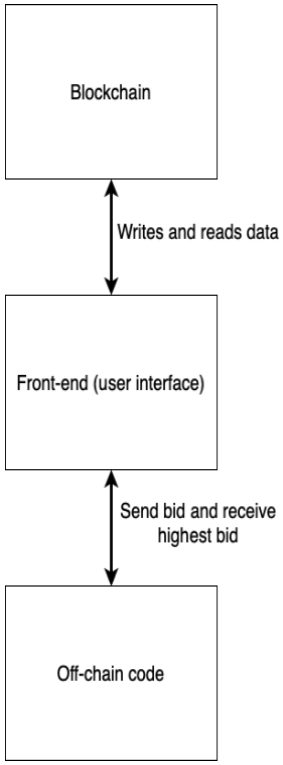

Figure 6. The main architecture of the system

As shown in figure 6, the system consists of three parts:

1. Blockchain: blockchain stores all necessary information regarding the auction. The users interact with blockchain to write or read any data. In other words, the blockchain is an immutable transparent database.
2. Front-end (user interface): provides a user-friendly environment for using the system. In addition, the front-end is responsible for the following tasks:
    - Off-chain code performance evaluation and verification as explained in table 2.
    - Winner information submission to blockchain.
    - Dispute request submission to blockchain.
3. Off-chain code: off-chain acts as the brain in this method. It receives the bids, calculates the highest value, and returns it to the users.

Note: All algorithms and pseudocodes are available in the appendix section.

3.2 Scenario 1 (No Cheating)

Scenario 1 explains the case when there is no fault in calculating the highest bid as illustrated in figure 7.

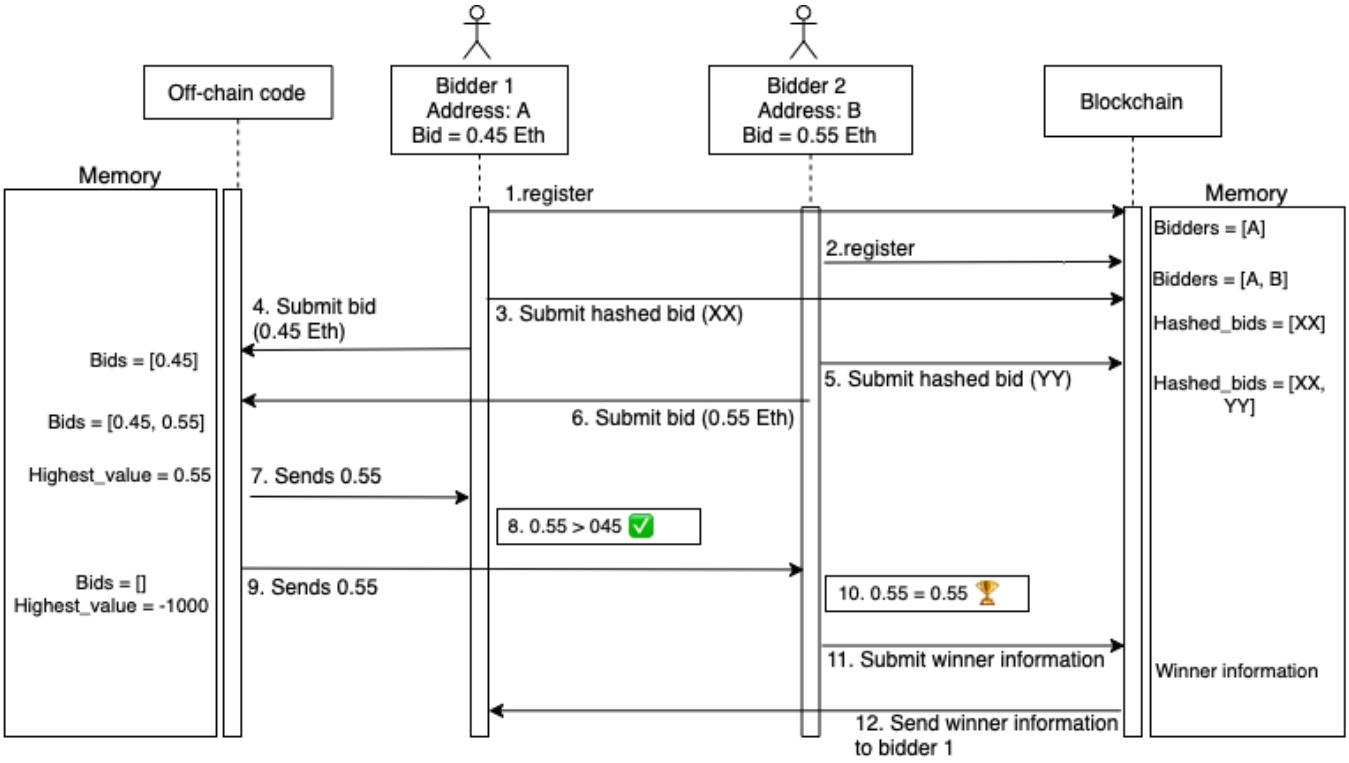

Figure 7. No cheating scenario

The steps for figure above are as follows:

1. User 1 with address A registers as a bidder in auction. Hence, the bidder's address is saved as a valid bidder in blockchain.
2. User 2 registers with address B as a bidder.
3. Bidder 1 submits the hash digest of the bid (XX) as computed in equation (1).

$$hashDigest = hash256(bid + nonce) \quad (1)$$

The input space referring to the bid is small due to the application logic (0 Ether to 1000 Ether). Malicious actor can conduct a reverse attack and find the input bid.
To address this, a random value named nonce (also known as randomness or salt) is concatenated with bid as shown in equation 1 to increase the input space and make the reverse attack difficult.

Submitting a hashed value into blockchain can bring the following advantages:

- The winner must disclose the bid and nonce. Having the message digest of the bid available on blockchain can force the winner to reveal the correct values and avoid any misbehaviors.
- Preventing bidders from any dishonest activity thanks to the deterministic feature of the hash functions explained in earlier chapter. Thereby, the bidders cannot modify the submitted bid.

4. Bidder 1 submits the bid value (0.45 Eth) into off-chain code. Off-chain code receives and keeps it in the bids array (bids = [0.45]).
5. Bidder 2 submits the hash of the suggesting bid (YY) into blockchain.
6. Bidder 2 sends the bid value (0.55 Eth) to off-chain code. Off-chain code stores it in bids array (bids = [0.45, 0.55]). Upon receiving the last bid, off-chain code calculates the highest value using equation 2. In this scenario, 0.55 is the maximum.

$$highestValue = max([bids]) \quad (2)$$

7. Off-chain code sends back the maximum (0.55 Eth) to bidder 1 for further evaluations.
8. Bidder 1 compares the maximum with their own value as shown in table 2. In this case, the maximum is greater than bid 1 (0.55 > 0.45). Hence, bidder 1 confirms the maximum value and waits until the winner is announced.
9. Off-chain code returns the maximum (0.55 Eth) to bidder 2. Upon sending the maximum value to the last bidder, off-chain code resets all the information for better privacy (highest_value = -1000 and bids = []).
10. Bidder 2 performs a comparison between maximum and their own bid. In this case, they are equal meaning that the bidder 2 is the winner.
11. Bidder 2 reveals the address, nonce, and bid to the blockchain as the winner.
    Therefore, in addition to the participants, blockchain users can then verify the auction.
12. Blockchain sends the winner information containing address, bid, and nonce to bidder 1.

3.3. Scenario 2 (Cheating)

The steps below explain the case when a bidder detects a fault in finding the highest bid as shown in figure 8.

1. Address A registers as a valid bidder (bidders = [A]).
2. Address B registers as a valid bidder (bidders = [A, B]).
3. Bidder 1 submits the hash (XX) into the blockchain using equation 1 (hashed_bids= [XX]). Scenario 1 explains in detail the reason for using hash and nonce.
4. Bidder 1 sends the bid (0.45 Eth) to off-chain code (bids = [0.45]).
5. Bidder 2 computes the hash (YY) following equation 1 and sends it to the blockchain (hashed_bids = [XX, YY].

6. Bidder 2 submits the bid (0.55) into off-chain code (bids = [0.45, 0.55). After that, off-chain computes the maximum of the existing bids incorrectly. In this scenario, the incorrect maximum is 0.45.
7. Off-chain code sends 0.45 as the maximum to bidder 1.
8. Bidder 1 realizes that the maximum is equal to their bid. This means the bidder 1 is the winner.
9. Bidder 1 sends the address, bid, and nonce as the winner to the blockchain.
10. Off-chain code sends the highest value to bidder 2.
11. Bidder 2 detects a fault in off-chain code. In the comparison, the bid belonging to bidder 2 is greater than the maximum. This means that there is a mistake in off-chain code and is successfully identified by one of the bidders.
12. Bidder 2 submits a dispute request into blockchain. By this request, bidder 2 must reveal the address, bid, and nonce. The blockchain users and auction participants can evaluate the request.
13. Blockchain sends the winner information to bidder 2.

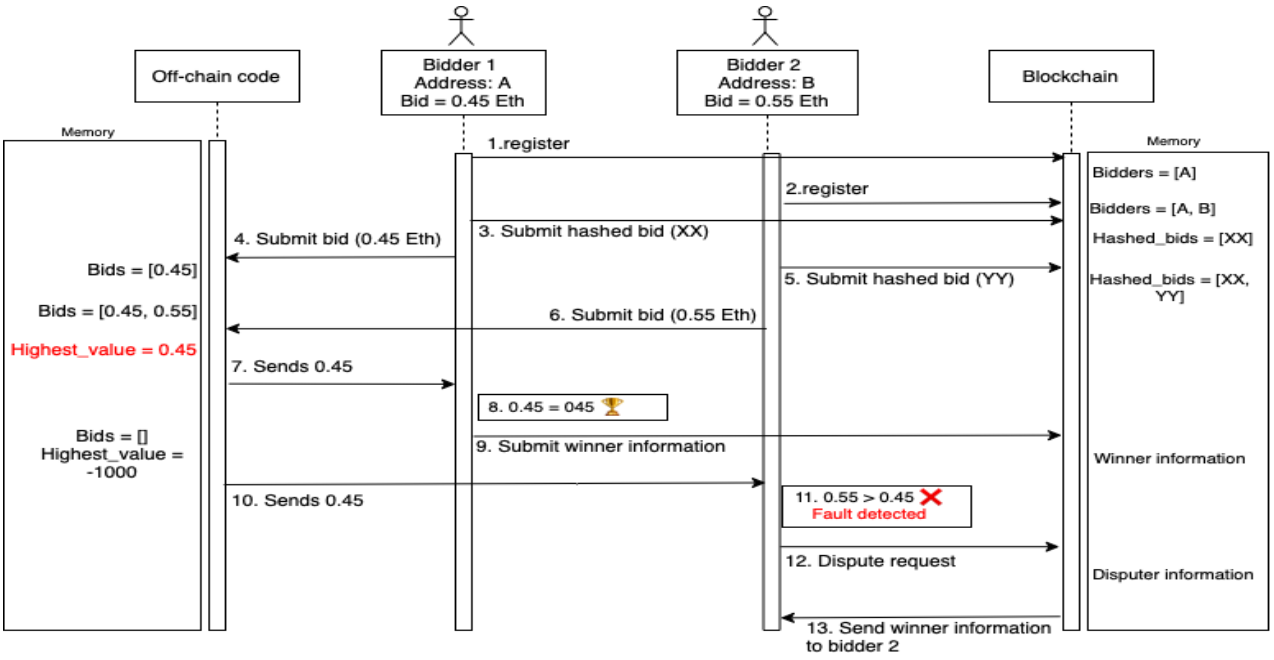

Figure 8. Cheating scenario

3.4 Flow Chart

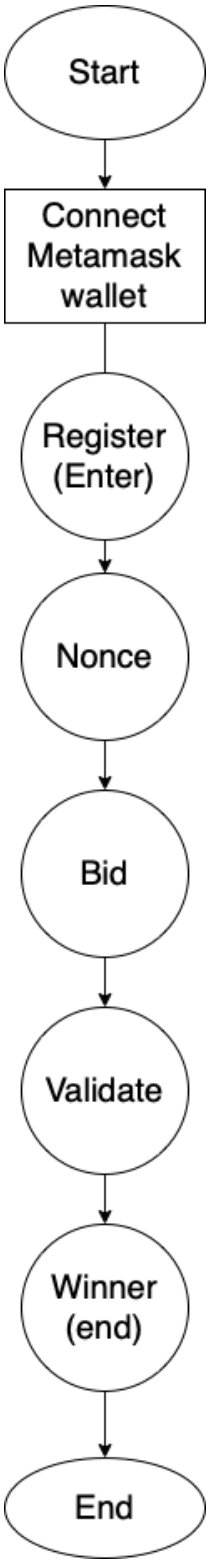

Figure 9. Main flow chart

3.5 GitHub

GitHub [27] is a popular platform allowing developers from different parts of the world to collaborate on software projects. it comes with discussion rooms, issue reporting panels, automated pipelines etc.

The proposed design repository link comes as follows:
https://github.com/AKEB-asyemmtric-key-each-bidder

3.6 Smart Contract (Blockchain)

3.6.1 Developed Code

Remix is the IDE and Solidity is the language used for developing the smart contract.

The developed code is available in this repository:
https://github.com/AKEB-asyemmtric-key-each-bidder/AKEB-SmartContract

The description of the functions is available in table 3.

Table 3. Description of smart contract functions

Function	**Description**
registerAuctionInfo	Seller calls this function to submit asset name and description.
getAuctionInfo	Getter function for bidders to observe the asset information.
registerBidder	Registering a user as a valid bidder.
getAllBidders	Returning a list of all bidders participating in the auction.
submitEncodedBid	Bidders submit the commitment of the bid.
getEncodedBid	Bidders who trigger this function receive their own submitted hash.
submitWinner	The winner submits the address, bid, and nonce by calling this function.
getAllWinners	This function returns a list of the winners. It is worth mentioning that it is likely to have multiple winners.

dispute	The bidder believing that there is a mistake in the progress calls dispute. By doing so, the bidder must reveal the bid and nonce for dispute evaluation.
getAllDisputers	The participants observe a list of users who have appealed.

3.6.2 Deploying Smart Contract

Remix provides a reliable environment for developing and testing the code. Once the code meets the desired expectations, it is ready for deployment.

Truffle, Infura, and Rinkeby test net are used for deployment:

1. Create a Truffle project by using the command below.

```
truffle init
```

2. Move the source code to a new file in “Contracts” folder.
3. Create a new migration file associated with the smart contract for a proper deployment process as shown in figure 10.

```
const AKEB = artifacts.require("AKEB");

module.exports = function (deployer) {
  deployer.deploy(AKEB);
};
```

Figure 10. Migration file of the smart contract

4. Create a new account in Infura to get API key and endpoint. Infura is required for deploying through Truffle.

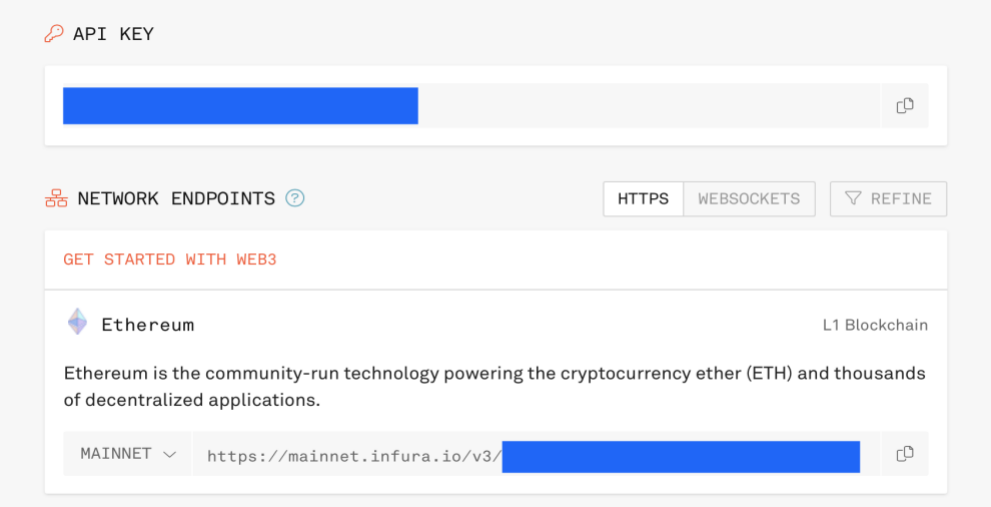

Figure 11. Infura dashboard

Note: API key is hidden in figure 11 for security purposes.

5. Install dotenv [28] package using the command below. Uploading sensitive information such as API keys into GitHub can bring serious security threats. To prevent this, developers can write this information into a file with ".env" extension. This file is ignored during the push into GitHub.

```
npm i dotenv
```

6. Place Infura API key and Metamask account private key into dotenv file.
7. Configure "truffle-config.js" file as shown in figure 12.

```
provider: () =>
  new HDWalletProvider({
    privateKeys: private_keys,
    providerOrUrl: `https://rinkeby.infura.io/v3/${rinkeby_key}`,
    numberOfAddresses: 1,
  }),
network_id: 4, // Rinkeby's id
gas: 5500000, // Rinkeby has a lower block limit than mainnet
confirmations: 2, // # of confs to wait between deployments.
// (default: 0)
timeoutBlocks: 200, // # of blocks before a deployment times out
// (minimum/default: 50)
skipDryRun: true, // Skip dry run before migrations
```

Figure 12. Code of deploying the smart contract into Rinkeby test net

8. Execute the command for deploying into Rinkeby.

```
truffle migrate -f 2 --network rinkeby
```

9. Smart contract is available in Etherscan with the following address: https://rinkeby.etherscan.io/address/0xa2b9a6507e8185ee652bb346034b5b47d581f0c4

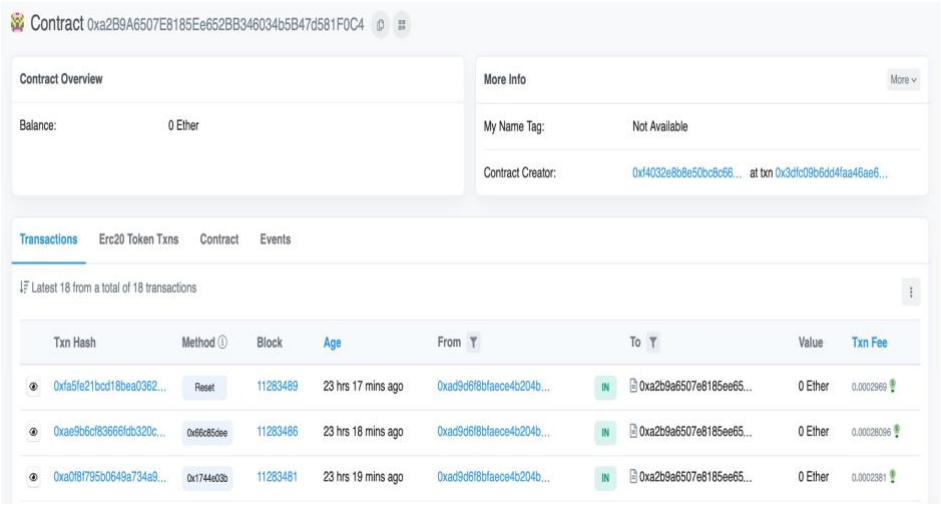

Figure 13. The smart contract page on Etherscan

10. Smart contract address is: 0xa2B9A6507E8185Ee652BB346034b5B47d581F0C4

3.7 Front-end (User Interface)

The front-end project is available in the following link:
https://github.com/AKEB-asyemmtric-key-each-bidder/Web-app

3.7.1 Reactjs

Reactjs [29] is a popular JavaScript framework for user interface development. Comparing to the pure JavaScript, HTML, and CSS, Reactjs brings the following benefits:

1. Reactjs projects consist of components reused throughout the code. Thereby, they bring reusability and reduce the size of the code.
2. It provides asynchronous API calls leading to a better user experience. “States” handle and show the data coming from external endpoints.
3. Reactjs provides an easy unidirectional way for components to transfer data.
4. Web applications are faster with Reactjs. While traditional web tools refresh all the components after a change, Reactjs does so only for the required ones.

3.7.2 Front-end Structure (React Tree)

The figure 14 shows the main components of the front-end design:

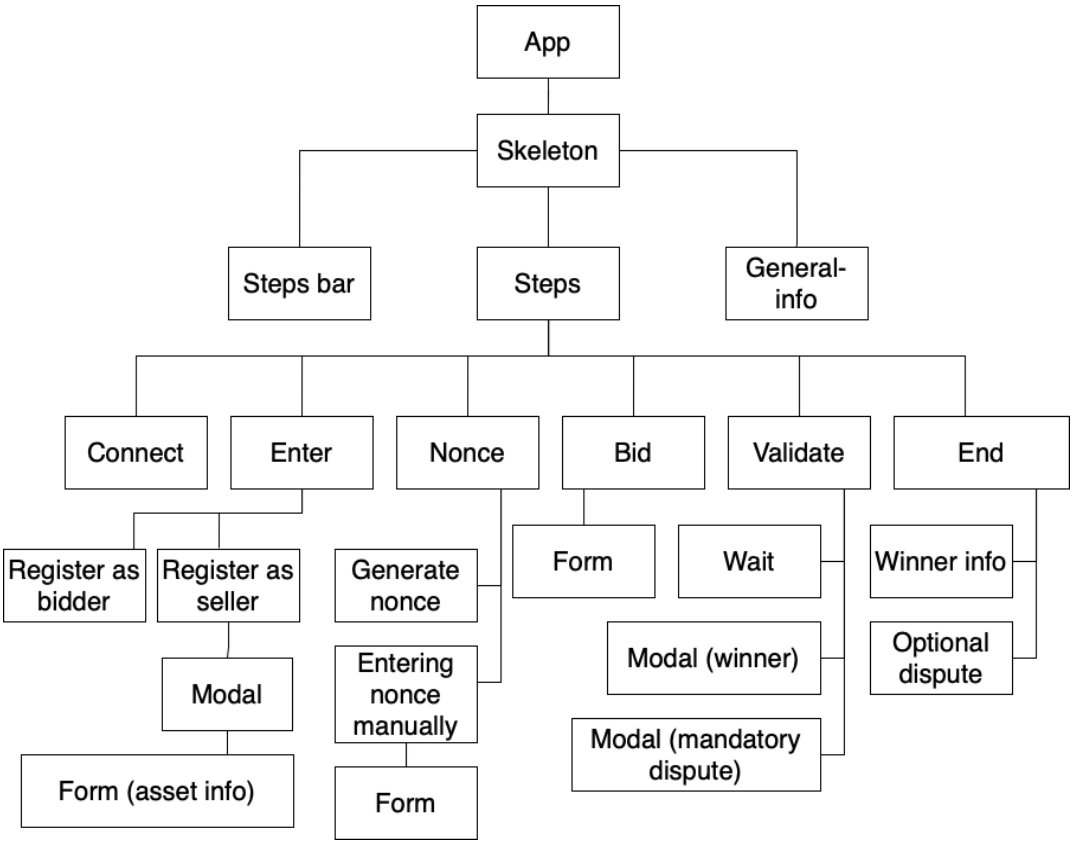

Figure 14. Front-end structure (React tree)

3.7.3 Dependencies and Packages

Some of the dependencies explained in table 4 are mandatory to use for Reactjs and some are optional as they follow the best practices and make the coding more straightforward.

Table 4. Dependency list of the front-end

Packages	**Version number**	**Explanation**
react	18.2.0	Reactjs package
axios	0.27.2	Package for https and http calls
web3	1.7.3	Used for interacting with blockchain networks
crypto-js	4.1.1	Provides trustable and secure cryptographic functions. This project only uses hash

		functions.
antd	4.21.4	This library provides beautiful built-in web components such as buttons, modals, alerts, and menus removing the need for developers to write boilerplate CSS and JavaScript codes.

3.7.4 Configuring Metamask

Metmask is a wallet for cryptocurrency exchange explained in 2.2.12. It is available as a browser extension (available for Google chrome, Brave, Microsoft edge, and Firefox). Users interested in interacting with blockchain-based web applications must install Metamask and transfer enough balance as shown in figure below.

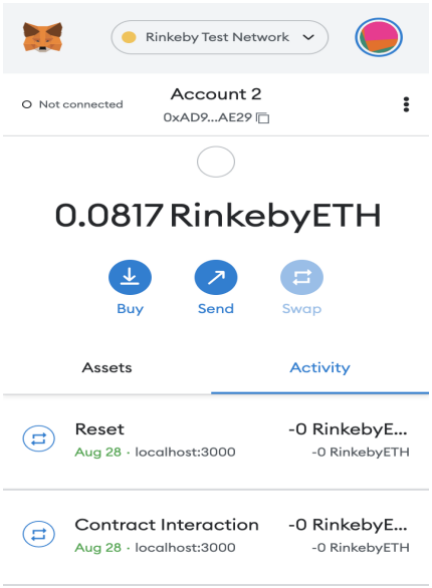

Figure 15. Accounts page of Metamask

Since this project is deployed into the Rinkeby test network, users can use Rinkeby faucet [30] to load their wallet with test Eth (RinkebyEth).

3.7.5 User Interface Skeleton

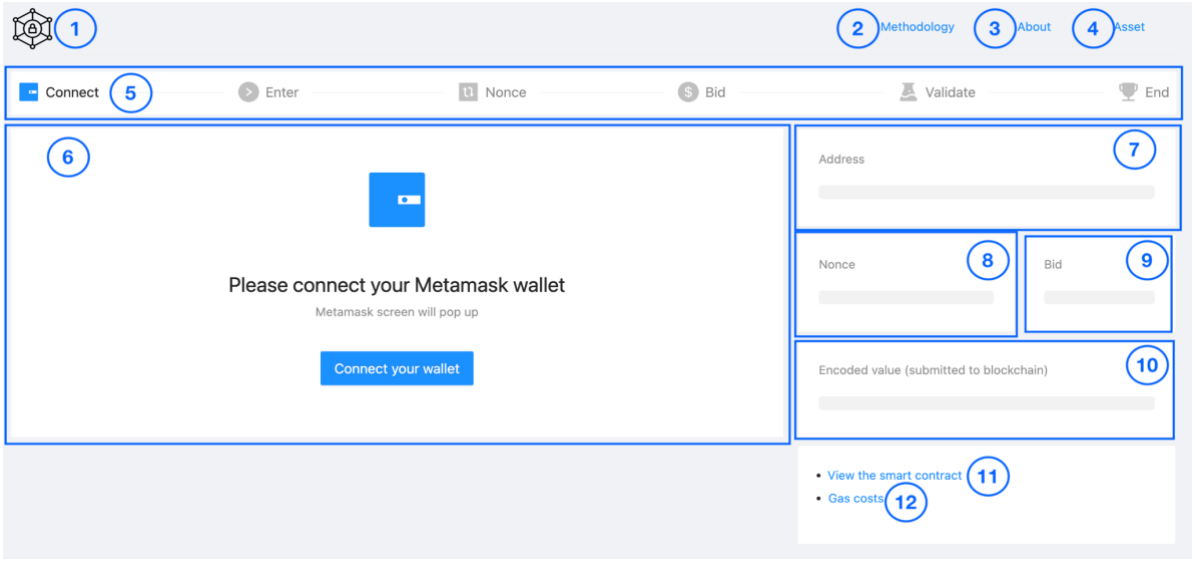

Figure 16. User interface skeleton

The skeleton of the designed user interface comes in figure 16. The highlighted sections in this figure are explained:

1. Logo: this is the logo for the proposed system (Designed by Parzival [31] from Flaticon [32]).
2. Methodology: it is a modal explaining the applied methodology as shown in figure 17.

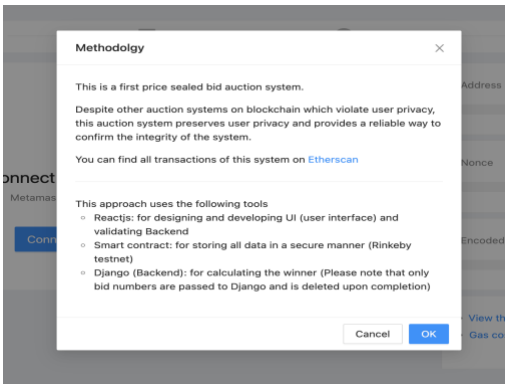

Figure 17. Methodology modal

3. About: by clicking, a modal appears introducing the developer as shown in figure 18.

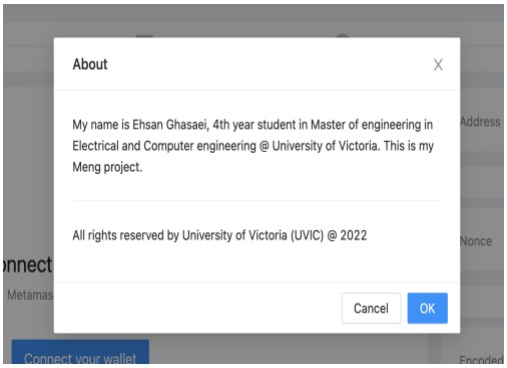

Figure 18. About modal

4. Asset: this modal provides information about the trading asset (figure 19).

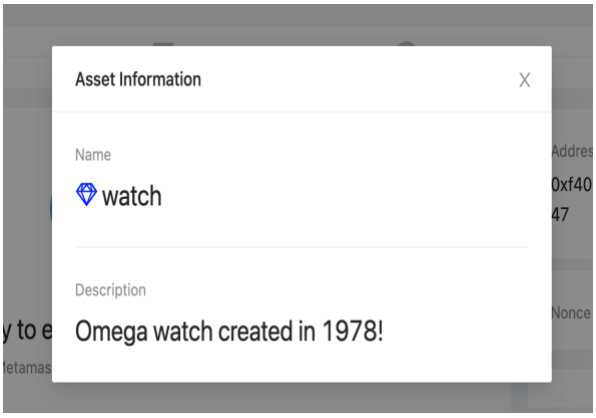

Figure 19. Modal presenting asset name and description

5. Step bar: it shows the past and current phases in the auction by turning their icons into blue.
6. Main action: it is the main component users interact with and changes according to the auction phase.
7. Address: it is the address of the participant.
8. Nonce: it is the nonce that either the system generates, or the bidder enters to increase the hash input space and make the guessing difficult.
9. Bid: the bid suggested by the user appears in this component.
10. Encoded value (submitted to blockchain): the hash output of the bid and nonce as shown in equation 1. This value is submitted to the blockchain.
11. View the smart contract: upon clicking this button, the user observes the smart contract page on Etherscan.
12. Gas cost: it is a modal listing all the gas costs corresponding to actions (figure 20).

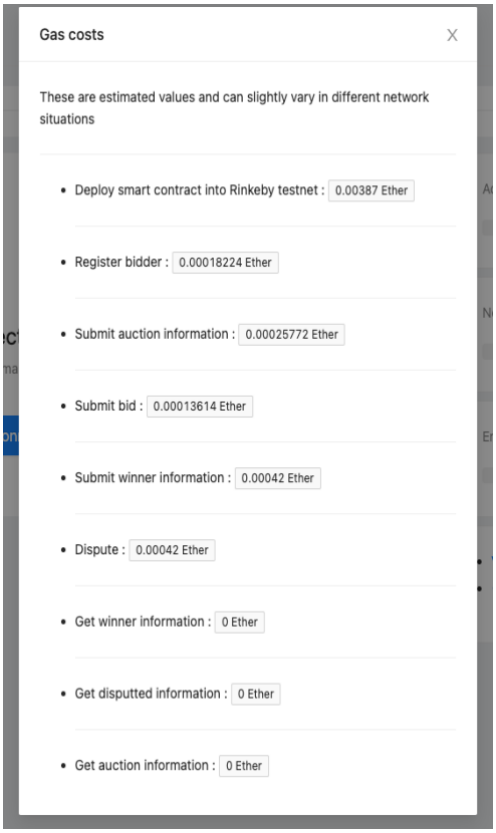

Figure 20. Gas cost modal

Note: Chapter 4 thoroughly describes the gas costs.

3.7.6 Connect Wallet Phase

Figure 21 shows the user interface for Metamask wallet connection. Upon clicking on the "Connect your wallet" button, the front-end receives the user's address from Metamask and shows it in the "Address" section.

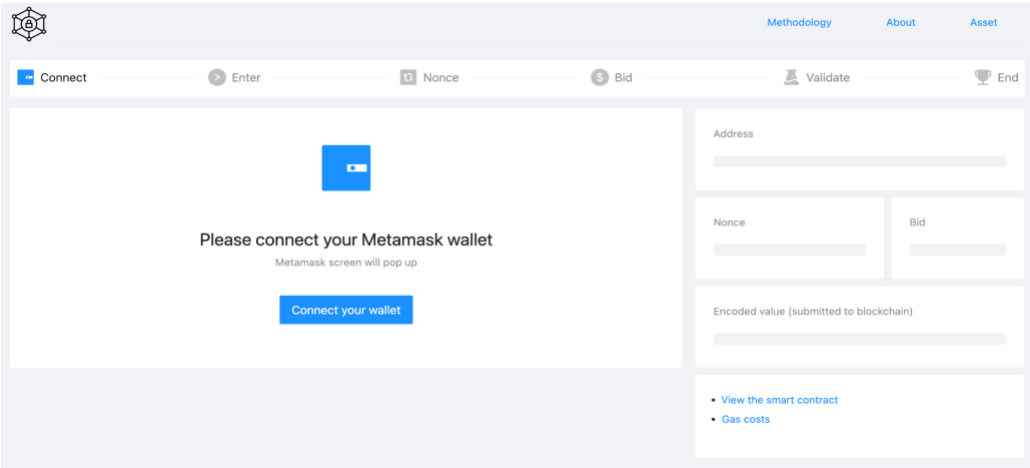

Figure 21. User interface of connect wallet phase

3.7.7 Enter (Register) Phase

Figure 22 illustrates the user interface for the enter phase. The users can act as a bidder or a seller.

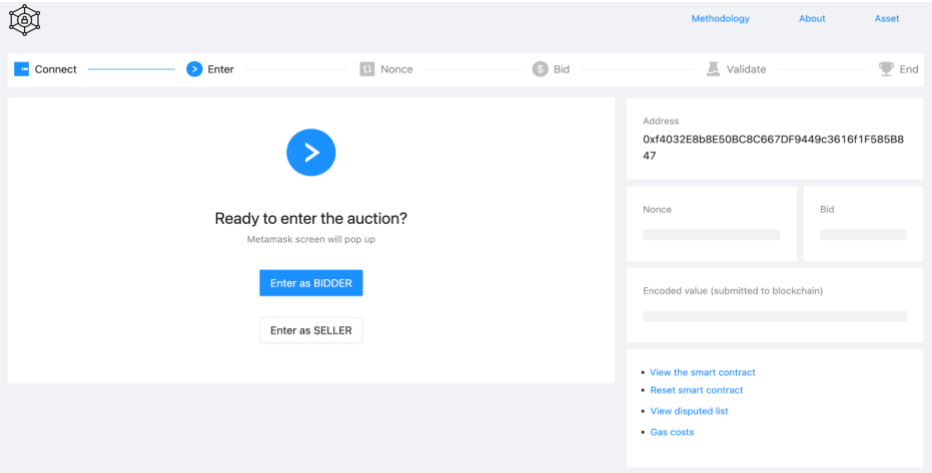

Figure 22. User interface of enter (register) phase

3.7.7.1 Flow Chart

The flow chart explaining the functionality of enter (register) phase for bidder and seller comes in figure 23.

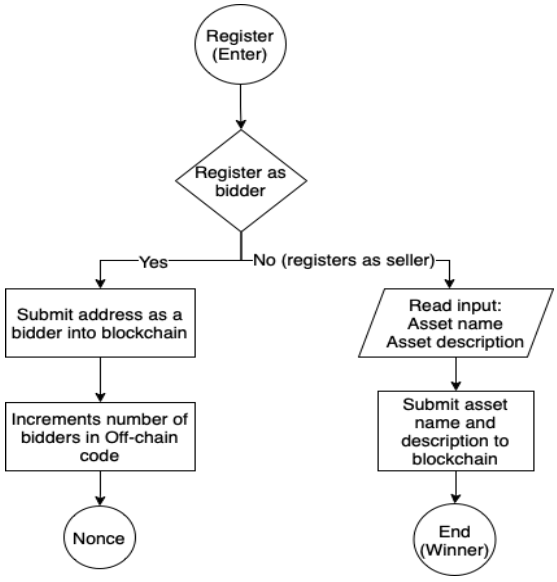

Figure 23. Flow chart of enter (register) phase

3.7.7.2 Enter as Bidder

When the bidders click on the “Enter as BIDDER” button, Metamask screen pops up to confirm the following transaction: submitting the bidder’s address into blockchain as shown in figure 24.

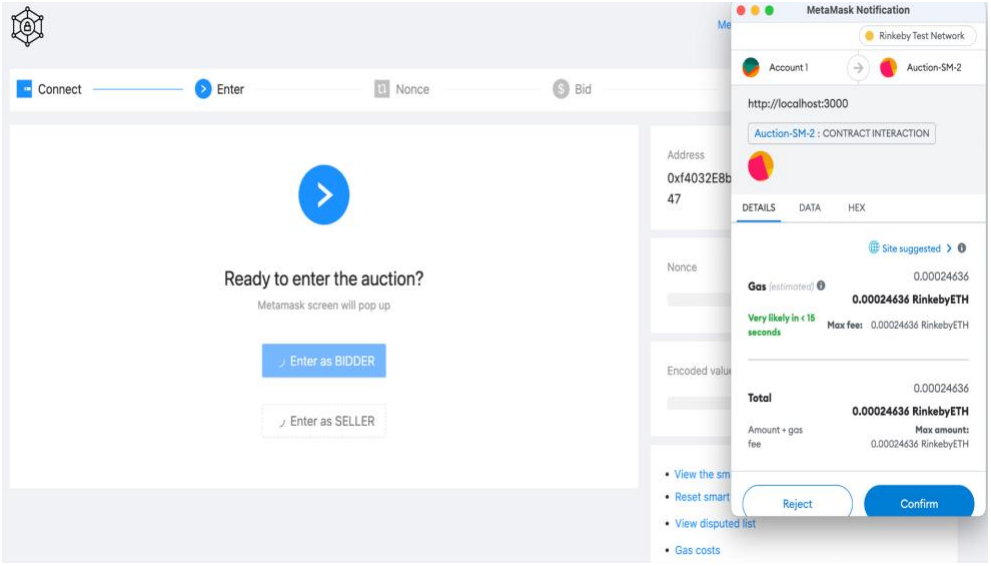

Figure 24. User interface of "enter as bidder"

3.7.7.3 Enter as Seller

Figure 25 is when the seller clicks on the “Enter as SELLER” button. In this case, a modal with a form appears on the screen to enter asset name and description. Once the seller completes the form and clicks on the “Submit” button, Metamask asks the seller to confirm the transaction.

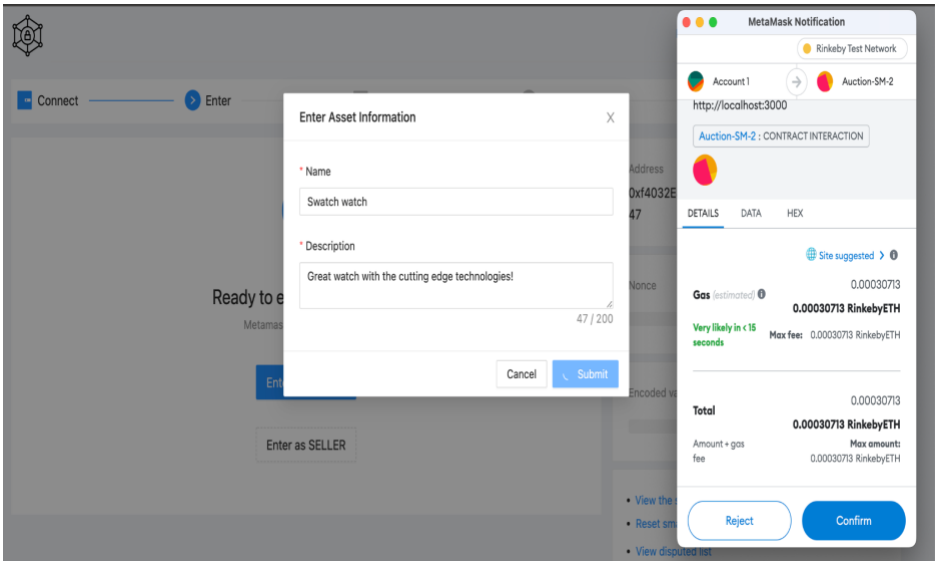

Figure 25. User interface of "enter as seller"

3.7.8 Nonce Phase

3.7.8.1 Flow Chart

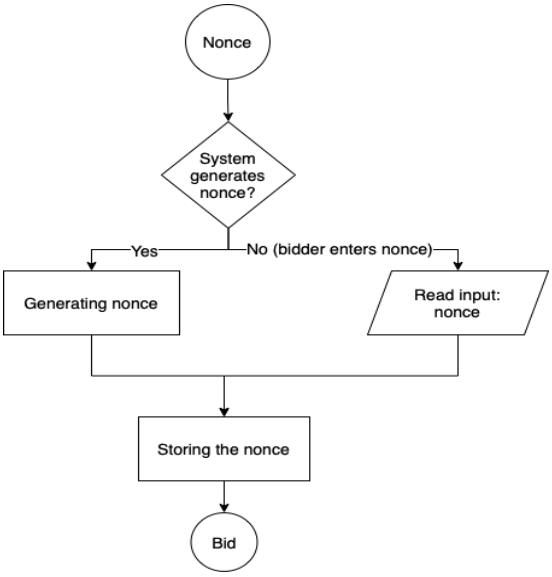

Figure 26. Flow chart of "Nonce" phase

3.7.8.2 System Generating the Nonce

The user has the option of allowing the system to generate the nonce as demonstrated in figure 27 or entering it manually.

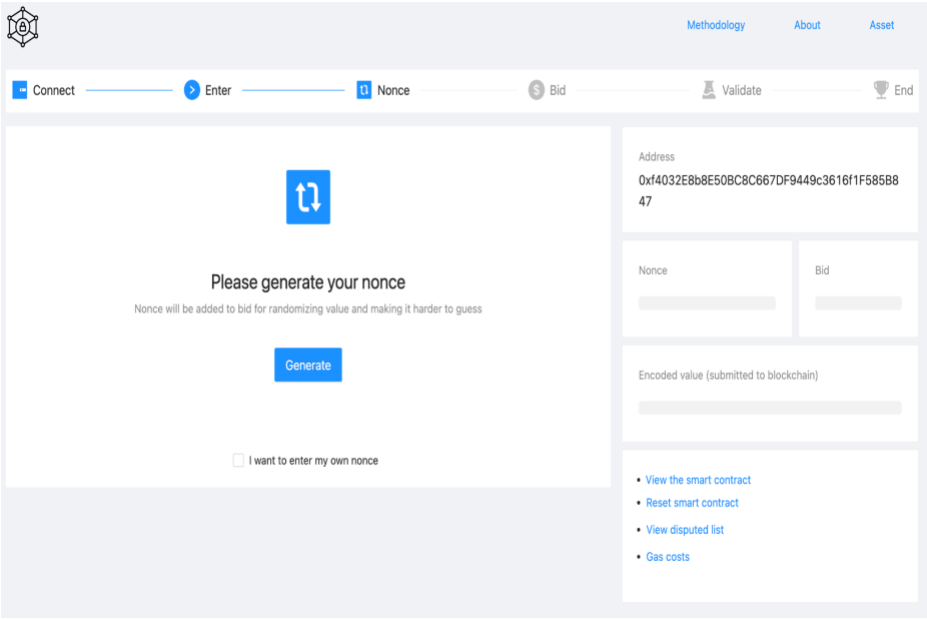

Figure 27. User interface of the case that system generates the nonce

3.7.8.3 User Entering Nonce Manually

Figure 28 is when the users prefer to enter the nonce manually.

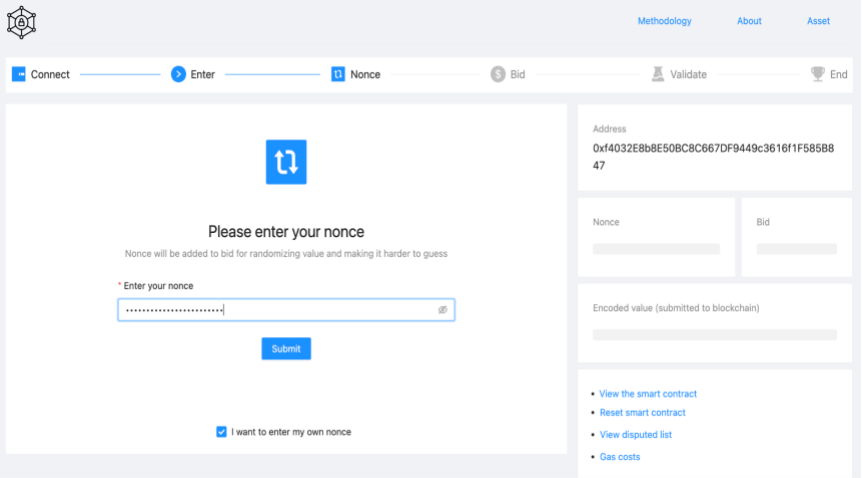

Figure 28. User interface of the case that user enters the nonce

3.7.9 Bid Phase

In this phase, the bidder enters the suggesting bid. Upon submission, the following occurs:

1. bid and nonce are concatenated and passed to the hash function as explained in 3.2.
2. The bidder submits the hash result into the blockchain.
3. The bidder sends the bid to off-chain code.

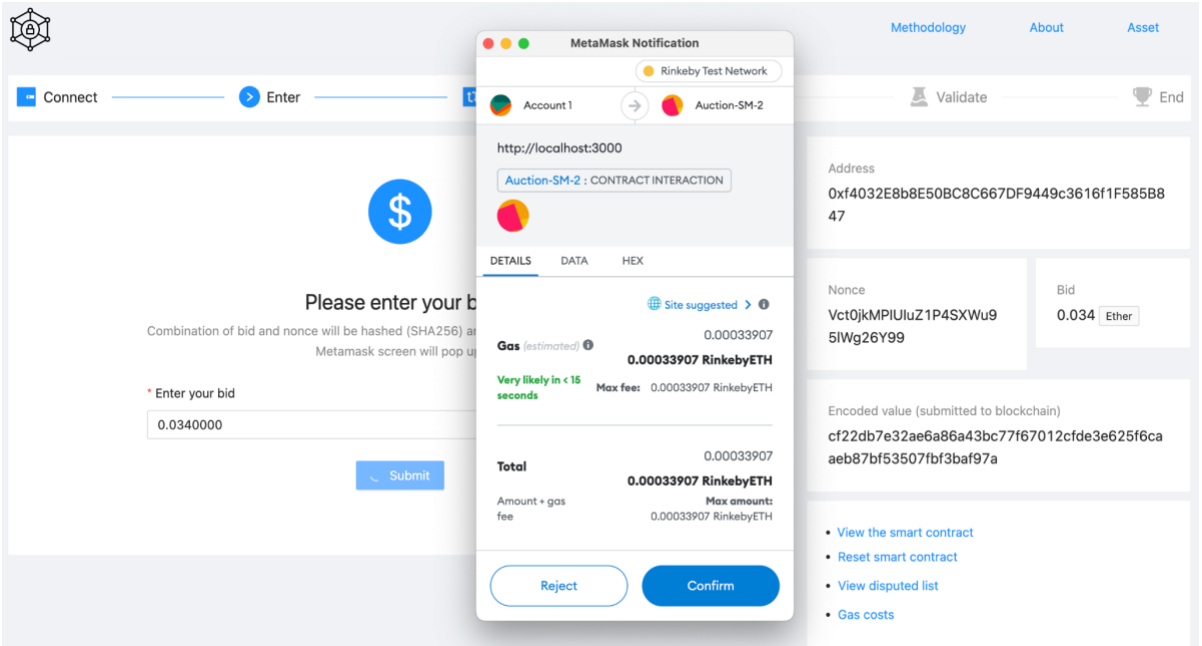

Figure 29. User interface of "Bid" phase

The value of bid, nonce, and hash are available in the corresponding components of figure 29 and table 5.

Table 5. The values of bid, nonce, and the hash in "Bid" phase

Name	**Value**
Nonce	Vct0jkMPIUIuZ1P4SXWu95lWg26Y99
Bid	0.034 Eth
Hash	cf22db7e32ae6a86a43bc77f67012cfde3e625f6caaeb87bf53507fbf3baf97a

3.7.9.1 Flow Chart

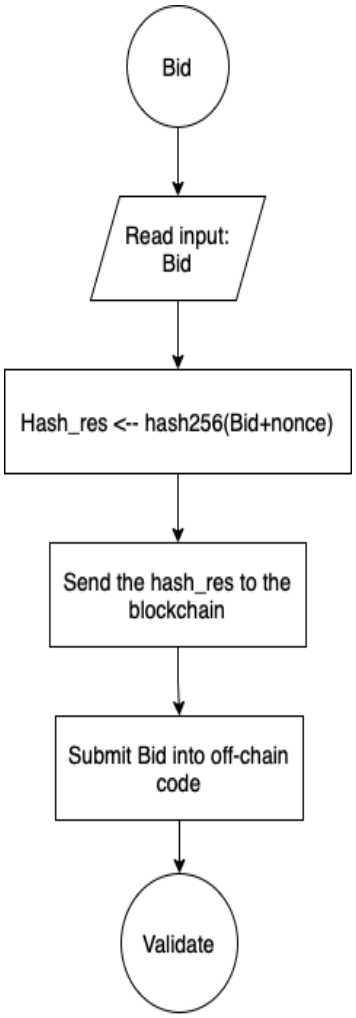

Figure 30. Flow chart of "Bid" phase

3.7.10 Validation Phase

Figure 31 is the validation phase. The bidder waits until off-chain code calculates and returns the maximum. Upon receiving the maximum, the front-end compares it with the bidder's value. The following cases might happen:

1. bid < maximum (confirmed): the bidder has confirmed the maximum and waits for the winner announcement.
2. bid == maximum (winner): the bidder is the winner and submits the winner information into the blockchain.
3. bid > maximum (dispute request): the bidder has detected a flaw in the system and appeals.

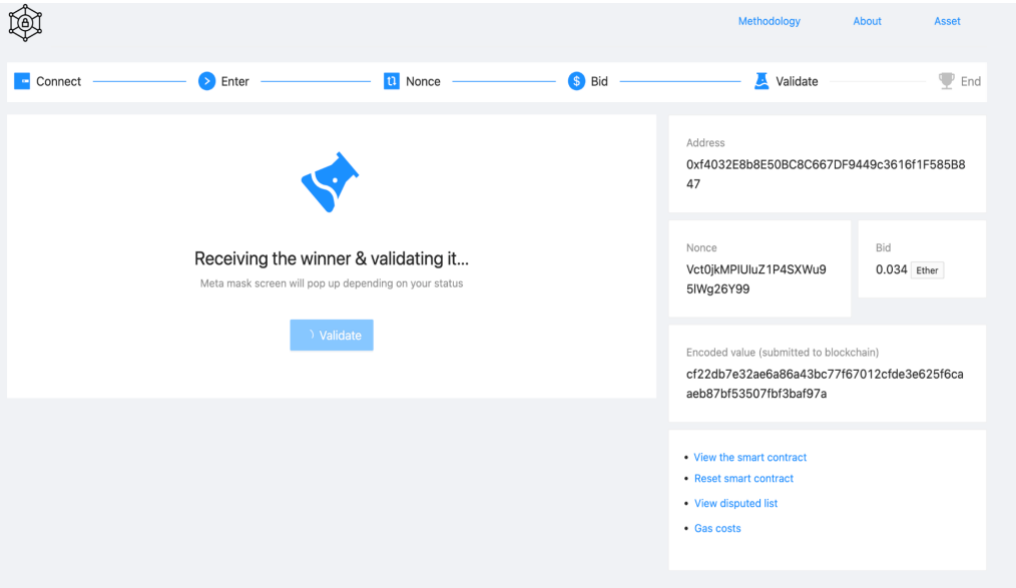

Figure 31. User interface of "Validation" phase

3.7.10.1 Flow Chart

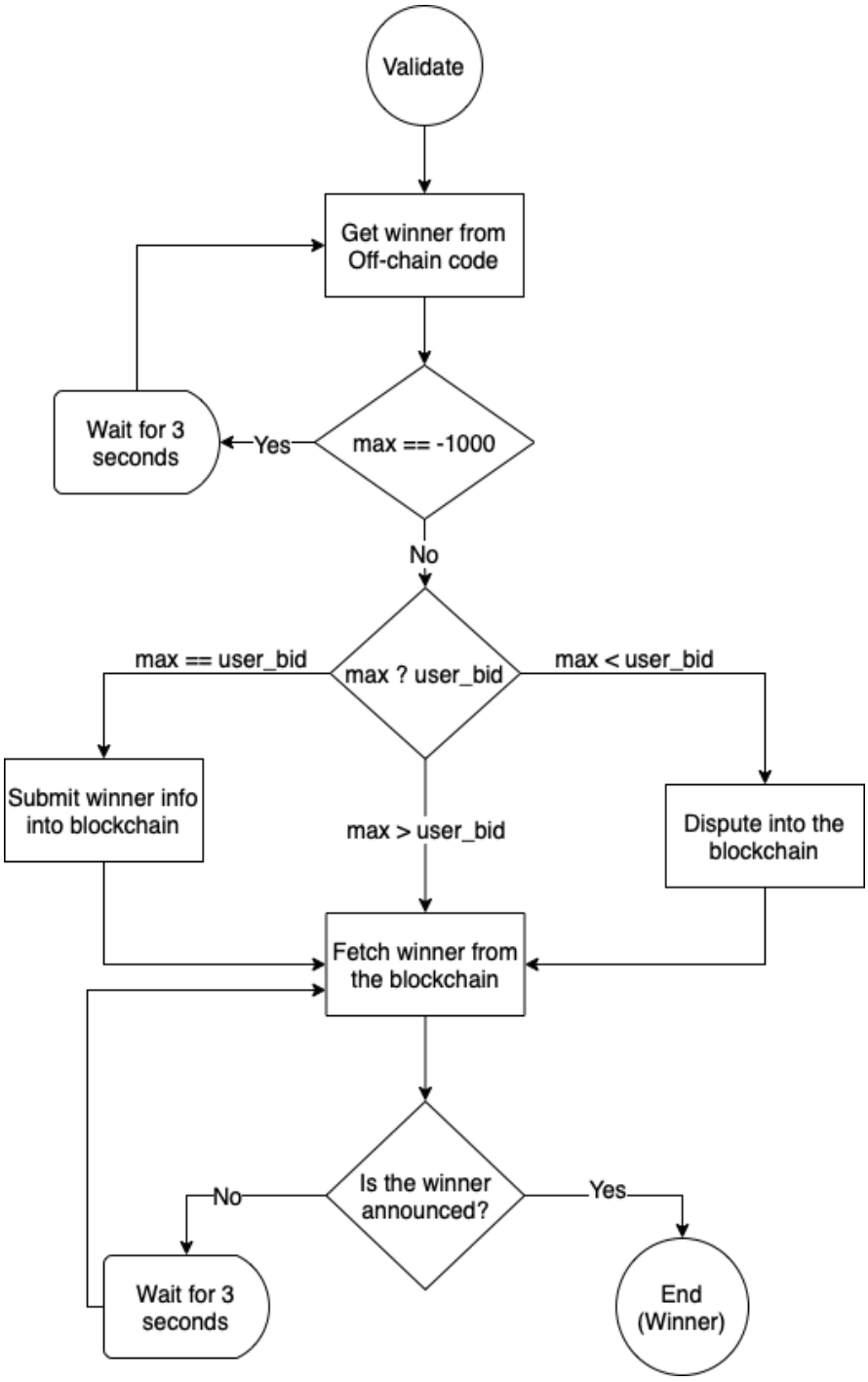

Figure 32. Flow chart of "Validation" phase

3.7.10.3 Neutral (Confirmed) Case

This is the case the bidder has successfully verified the computed maximum.

3.7.10.4 Winner Case

In the winner case, a modal appears describing the next steps and asking to submit the winner information as shown in figure 33.

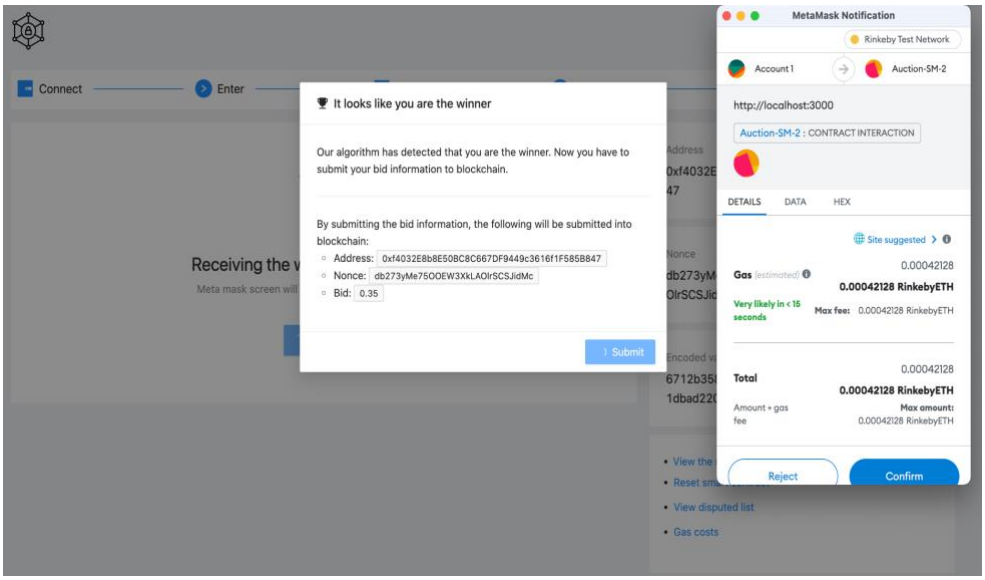

Figure 33. User interface of winner case

3.7.10.5 Dispute Request Case

The front-end shows the dispute modal if it detects a problem in the system. This modal explains the situation and asks the user to dispute (figure 34).

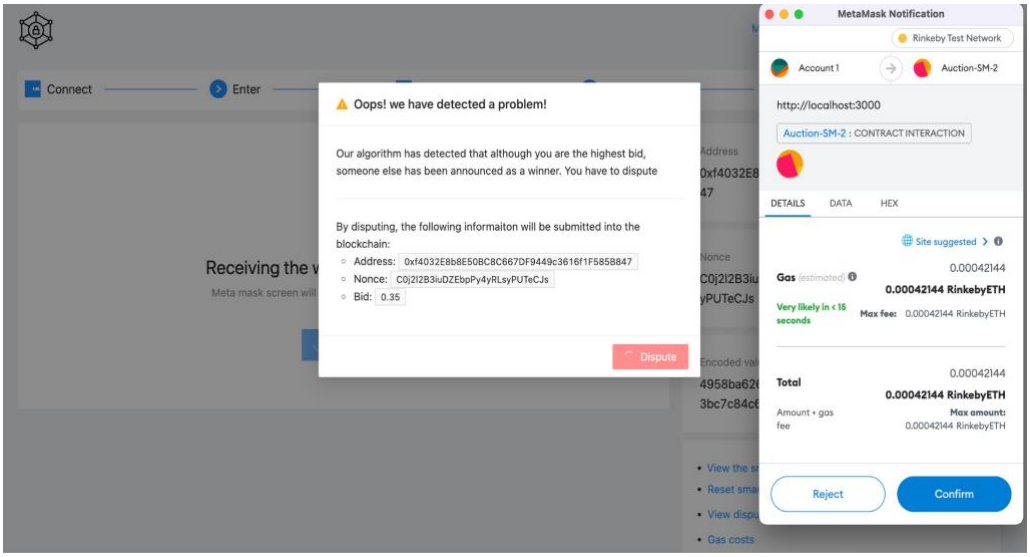

Figure 34. User interface of "Dispute Request" case

3.7.11 Winner (End) Phase

This phase shows the address, bid, and nonce of the winner. The user also can manually dispute by clicking “Dispute button” or refresh the winners list by clicking on refresh button.

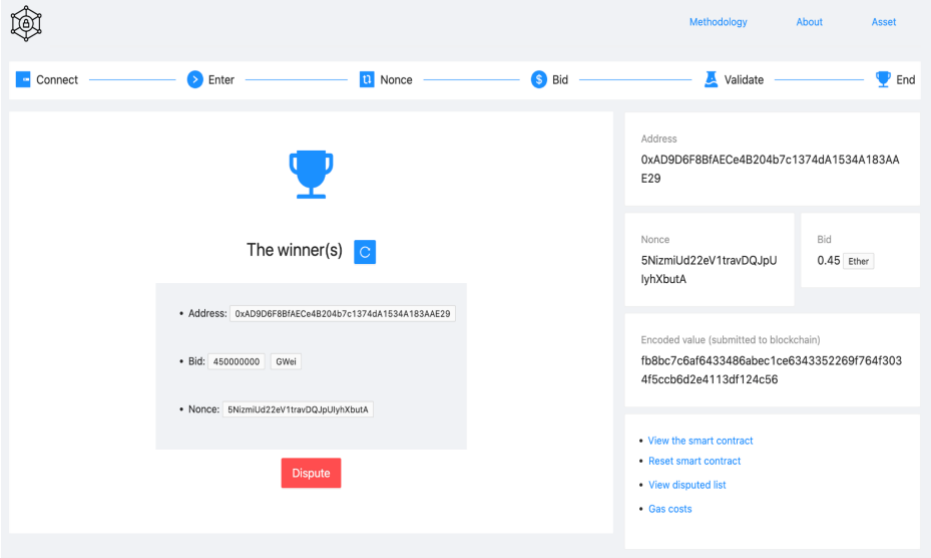

Figure 35. User interface of "Winner (End)" phase

3.7.11.1 Flow Chart

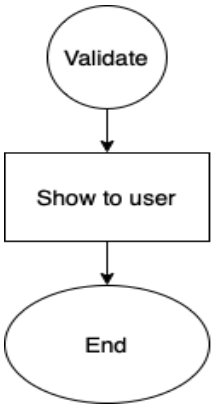

Figure 36. Flow chart of "Winner (End)" phase

3.8 Off-chain Code

3.8.1 Django

Django is a web framework available in Python for back-end development [33]. It is cleanly structured, straight forward, and follows MVC [33]. MVC classifies the project into model, view, and controller. Doing so helps the structure to stay organized. Also, it is easier for developers to implement and maintain the code.

3.8.2 API

Sometimes, software requires communication with external resources. API (application programming interface) fulfills this need. In the proposed design, Django API consists of GET and POST requests initiated from the front-end to trigger off-chain code. “rest_framework” is the desired dependency to be installed in Django projects.

3.8.3 Architecture

According to figure 37, off-chain code performs the following actions upon receiving a request from front-end:

1. Front-end (user interface) initiates the request.
2. urls.py handles all the requests coming from external resources and passes to appropriate functions in views.py.
3. views.py triggers the controller to perform specific actions.
4. The model (data.json) stores the information regarding the auction such as bids, number of bidders, and winner. The only component modifying the model is the controller. This alteration can be incrementing the number of bidders, bid submission, resetting the data etc.
5. Once the data is successfully changed, the response including optional data is passed to views.py.
6. Views.py adds the data coming from the controller into the HTTP response object and passes to urls.py.
7. Urls.py returns the response to the front-end.

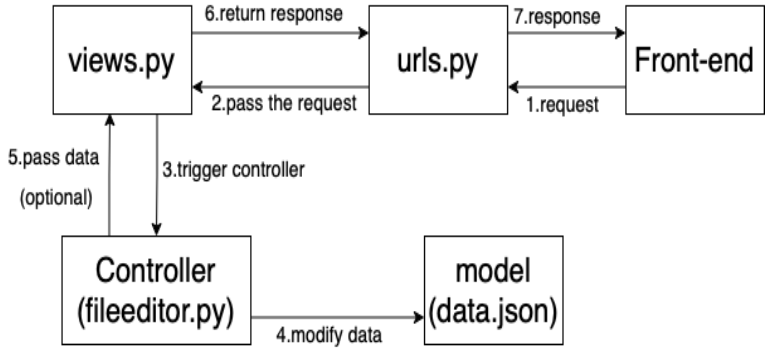

Figure 37. Off-chain code architecture

3.8.4 Endpoints

Note: further information about the endpoints is available in the appendix.

3.8.4.1 Increment the Number of Bidders

This is the first endpoint for the front end to communicate with the off-chain code. By calling this endpoint, the number of bidders on off-chain is incremented. Keeping this value helps off-chain code to find out whether all the bidders have submitted their bids or not. If so, it calculates the highest value.

3.8.4.2 Submit Bid

Bidders call the second endpoint for submitting their bids into off-chain code. Off-chain code collects the bids and finds the maximum at the appropriate time.

3.8.4.3 Get Winner

The last endpoint used to return the maximum to the clients. Once all the bidders receive this value, the data in the model (data.json) is deleted for privacy reasons.

Chapter 4: Performance and Cost Analysis

4.1 Privacy

Concealing the bid in the blockchain network while ability to evaluate the integrity is the main objective of the proposed methodology. As discussed in earlier chapters, instead of submitting the bids in plaintext, the hash digests are submitted into Rinkeby test network as shown in figure 38. Rinkeby and public users and bidders can only observe the hashes and cannot find the bids since the hashes are resistant against reverse attack.

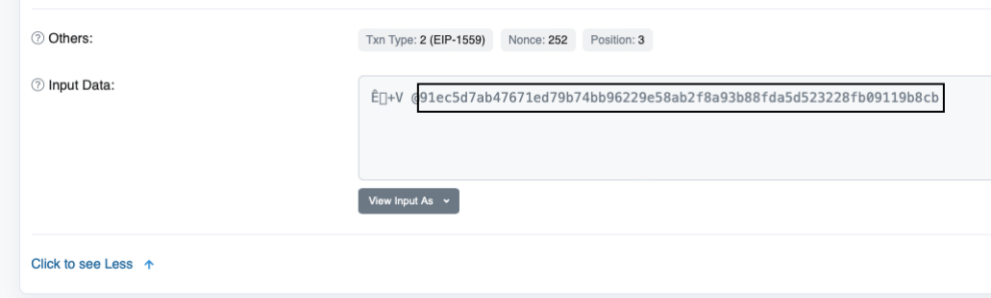

Figure 38. Hash is submitted into Rinkeby

According to the system design, the winner and disputers must reveal the bids and nonce for verification purposes. The winner information is used for auction validation, disputing in this design has the cost of disclosing the bid and nonce. By this mechanism, the bidders avoid unnecessary dispute requests. Thereby, the system helps preserving privacy for the losing bids.

Once off-chain code completes the tasks, it deletes all data in the model for preventing any unauthorized access and enhancing privacy (explained in 3.8.4.3).

4.2 Cost

4.2.1 Smart Contract

Performing actions on the blockchain requires paying the fee. This fee can vary based on the action type.
Tables 6 and 7 compare the required gas costs in USD to execute an auction between the proposed methodology and [7].

Note: the gas cost might slightly vary based on the blockchain network situation.
Note: At the time of this study, 1 Eth is approximately 1574 US dollars.

Table 6. The cost of the smart contract

Operation	**Cost (Eth)**	**Cost (USD)**
Contract deployment	0.00362	5.69 $
Register as a bidder	0.00018	0.28 $
Submit asset information	0.00026	0.41 $
Bidding	0.00013	0.20 $
Submit winner	0.00042	0.66 $
Dispute	0.00042	0.66 $
Get winner	0	0 $
Get dispute list	0	0 $
Get asset information	0	0 $

Table 7. The cost of [7]

Operation	**Cost (Eth)**	**Cost (USD)**
Contract deployment	0.00072	1.13 $
Bidding	0.00008	0.12 $
Bid opening	0.00007	0.11 $
End the auction	0.00005	0.07 $

Comparing the common operations, the proposed solution is slightly more expensive than [7]. For example, in deploying the contract, the gas fee referring to the proposed methodology is higher since it provides setting and getting asset information, accepting multiple winners, registering bidder etc. Regarding bidding, the costs are approximately the same.

4.2.2 Front-end and Off-chain Code

The front-end and off-chain code need the deployment into a server to be accessible by the bidders. Table 8 shows the cost in USD for deploying front-end and off-chain code by different providers per month [36]:

Table 8. The cost of the front-end and off-chain code deployment per month

Digital Ocean	**Google Cloud**	**Amazon Web Service**
5 $	7 $	9 $

According to the table above, Digital Ocean has the most reasonable price.

4.3 Vulnerabilities

A software company implements and deploys the off-chain code to the cloud, as explained in 4.2.2. One of the common vulnerabilities is unauthorized access to “data.json” resulting in bids leakage.

The following approaches can address the privacy concern as future works:

1. Encrypting the content of “data.json” in such a way that only the controller can decrypt and access data.
2. Recording the activities on “data.json” file, such as file opening and modification, through the auction and sending the record information to the blockchain for transparency and validation.
3. The off-chain code can compute the maximum once a bid is received. This approach removes the necessity for storing the offers and improves privacy.

Chapter 5: Conclusion and Future Work

The proposed approach in this research helps resolving the privacy issue in blockchain-based auction systems. The system consists of a front-end, blockchain network, and off-chain code. The bidders submit the hash of the bids into the blockchain and send the bid to the off-chain code. Then, the off-chain code calculates the maximum bid. In the next step, the maximum bid value is validated by each and every bidder. If each of the participants identifies a flaw in the system, they will dispute by revealing their bids and the nonce value. Lastly, the winner submits the bid and nonce values into the blockchain for verification purposes. Therefore, the proposed method helps preserving privacy for losing bids.

The designed system has multiple implemented features such as bidders' registration, nonce generation or entrance, hash generation and submission, winner verification, dispute request, acquiring the winner's information, obtaining the asset name and description, and viewing the list of disputed users. Also, the seller can enter the asset information and get the corresponding winner's information.

In bid submission, the proposed application costs almost the same as the existing blockchain-based auctions. In the proposed approach, the cost of the smart contract deployment is higher since the smart contract provides more functionalities. In addition, the front-end and the off-chain deployments bring costs.

In future work, the smart contract code can be refactored to optimize the execution cost. For instance, hash algorithms with smaller output space can be used to reduce the cost of submitting a bid. Moreover, the smart contract can be further improved in such a way that the winner and seller can exchange Eth and token. In addition, the privacy of the off-chain code can be enhanced as explained in 4.3.

References

[1] Chen, Biwen, Xue Li, Tao Xiang, and Peng Wang. "SBRAC: Blockchain-based sealed-bid auction with bidding price privacy and public verifiability." *Journal of Information Security and Applications* 65 (2022): 103082.

[2] Chen, Yi-Hui, Shih-Hsin Chen, and Iuon-Chang Lin. "Blockchain based smart contract for bidding system." In *2018 IEEE International Conference on Applied System Invention (ICASI)*, pp. 208-211. IEEE, 2018.

[3] Desai, Harsh, Murat Kantarcioglu, and Lalana Kagal. "A hybrid blockchain architecture for privacy-enabled and accountable auctions." In *2019 IEEE International Conference on Blockchain (Blockchain)*, pp. 34-43. IEEE, 2019.

[4] Galal, Hisham S., and Amr M. Youssef. "Succinctly verifiable sealed-bid auction smart contract." In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pp. 3-19. Springer, Cham, 2018.

[5] Chen, Yi-Hui, Li-Chin Huang, Iuon-Chang Lin, and Min-Shiang Hwang. "Research on blockchain technologies in bidding systems." *International Journal of Network Security* 22, no. 6 (2020): 897-904.

[6] "Vickrey Auction." 2020. Corporate Finance Institute. May 12, 2020. https://corporatefinanceinstitute.com/resources/knowledge/other/vickrey-auction/.

[7] Braghin, Chiara, Stelvio Cimato, Ernesto Damiani, and Michael Baronchelli. "Designing smart-contract based auctions." In *International conference on security with intelligent computing and big-data services*, pp. 54-64. Springer, Cham, 2018.

[8] Frankenfield, Jake. 2016. "Ethereum." Investopedia. February 24, 2016. https://www.investopedia.com/terms/e/ethereum.asp.

[9] "Transactions." n.d. Ethereum.org. Accessed September 9, 2022. https://ethereum.org/en/developers/docs/transactions/.

[10] "Cryptography Hash Functions." n.d. Tutorialspoint.com. Accessed September 9, 2022. https://www.tutorialspoint.com/cryptography/cryptography_hash_functions.htm.

[11] Vermaak, Werner. 2021. "What Are Peer-to-Peer (P2P) Networks?" CoinMarketCap Alexandria. CoinMarketCap. January 26, 2021. https://coinmarketcap.com/alexandria/article/what-is-peer-to-peer-p2p.

[12] "Proof-of-Work (PoW)." n.d. Ethereum.org. Accessed September 9, 2022. https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/.

[13] "Ethereum Virtual Machine (EVM)." n.d. Ethereum.org. Accessed September 9, 2022. https://ethereum.org/en/developers/docs/evm/.

[14] "Ethereum Accounts." n.d. Ethereum.org. Accessed September 9, 2022. https://ethereum.org/en/developers/docs/accounts/.

[15] "Introduction to Smart Contracts." n.d. Ethereum.org. Accessed September 9, 2022. https://ethereum.org/en/developers/docs/smart-contracts/.

[16] Varela-Vaca, Ángel Jesús, and Antonia M. Reina Quintero. 2022. “Smart Contract Languages: A Multivocal Mapping Study.” *ACM Computing Surveys* 54 (1): 1–38. https://doi.org/10.1145/3423166.
[17] “Remix - Ethereum IDE.” n.d. Ethereum.org. Accessed September 9, 2022. https://remix.ethereum.org.
[18] Blog, Moralis. 2021. “Remix Explained - What Is Remix? - Moralis Web3.” Moralis Web3 | Enterprise-Grade Web3 APIs. August 20, 2021. https://moralis.io/remix-explained-what-is-remix/.
[19] “The Crypto Wallet for Defi, Web3 Dapps and NFTs.” n.d. Metamask.Io. Accessed September 9, 2022. https://metamask.io.
[20] “Home - Truffle Suite.” n.d. Trufflesuite.com. Accessed September 9, 2022. https://trufflesuite.com.
[21] Blog, Moralis. 2021. “Truffle Explained - What Is the Truffle Suite? - Moralis Web3.” Moralis Web3 | Enterprise-Grade Web3 APIs. July 14, 2021. https://moralis.io/truffle-explained-what-is-the-truffle-suite/.
[22] “Mocha - the Fun, Simple, Flexible JavaScript Test Framework.” n.d. Mochajs.org. Accessed September 9, 2022. https://mochajs.org.
[23] “Ethereum API.” n.d. Infura. Accessed September 9, 2022. https://infura.io.
[24] “Frequently Asked Questions.” n.d. Infura. Accessed September 9, 2022. https://infura.io/faq/general.
[25] N.d. Etherscan.Io. Accessed September 9, 2022. https://rinkeby.etherscan.io.
[26] N.d. Etherscan.Io. Accessed September 9, 2022. https://etherscan.io.
[27] *GitHub: Where the World Builds Software*. n.d. Accessed September 9, 2022.
[28] Mot. n.d. *Dotenv: Loads Environment Variables from .Env for Nodejs Projects*. Accessed September 9, 2022.
[29] “React.” n.d. Reactjs.org. Accessed September 9, 2022. https://reactjs.org.
[30] N.d. Rinkebyfaucet.com. Accessed September 9, 2022. https://rinkebyfaucet.com/.
[31] Parzival’. n.d. “Free Icons Designed by Parzival’ 1997.” Flaticon. Accessed September 9, 2022. https://www.flaticon.com/authors/parzival-1997.
[32] “Free Icons and Stickers - Millions of Resources to Download.” n.d. Flaticon. Accessed September 9, 2022. https://www.flaticon.com.
[33] “The Web Framework for Perfectionists with Deadlines.” n.d. Djangoproject.com. Accessed September 9, 2022. https://www.djangoproject.com.
[34] Christie, Tom. n.d. “Home - Django REST Framework.” Django-rest-framework.org. Accessed September 9, 2022. https://www.django-rest-framework.org.
[35] N.d. Amazon.com. Accessed September 9, 2022. https://aws.amazon.com/what-is/api/.
[36] “DigitalOcean – The Developer Cloud.” n.d. Digitalocean.com. Accessed September 9, 2022. https://www.digitalocean.com/?refcode=7d9a2c75356d&utm_campaign=Referral_Invite&utm_medium=Referral_Program&utm_source=CopyPaste.

Appendix

Appendix 1: Reverse Attack

Input: submitted_hashes;

```
function reverseAttack():
    for i in range 0 to 1000 with step 0.001:
        if hash_256(i) == submitted_hashes:
            print("Input bid is found : ", i)
            break
```

Appendix 2: Verify Auction

Input:
submitted_hashes;
winnerNonce;
winnerBid;

```
function verifyAuction():
    isWinnerHonest = false
    for hash in submitted_hashes:
      if hash_256(winnerNonce + winnerBid) == hash:
          isWinnerHonest = true
          print("Winner has revealed the real bid and nonce")
          break

    if isWinnerHonest == false
      print("Winner has revealed the incorrect bid and nonce"
```

Appendix 3: Verify Dispute

Inputs:
submitted_hashes;
winnerNonce;
winnerBid;
disputerBid;
disputerNonce;

```
function verifyDispute():
    isDisputeAccepted = false
    for hash in submitted_hashes:
      if hash_256(disputerNonce + disputerBid) == hash:
          if disputerBid > winnerBid:
            isDisputeAccepted = true
            print("Dispute accepted")
            break
```

```
if isDisputeAccepted == false
   print("Dispute rejected")
```

Appendix 4: Connect Wallet

Inputs:
Using Button: connectWallet

connectWallet:
```
web3 = browserWindow.ethereum.request("accounts");
accounts = web3.requestAccounts()
address = accounts[0].address
AddressText.showText(address)
contract = createSmartContract(web3)
goToRegister()
```

createSmartContract(web3)
```
address = "0xa2B9A6507E8185Ee652BB346034b5B47d581F0C4"
abi available
contract = web3.eth.Contract(abi, address)
return contract
```

Appendix 5: Enter as Bidder

Inputs:
Using "Enter as BIDDER" Button: registerBidder

registerBidder
```
registerOnBlockchain
registerOnOffChainCode
goToNonce()
```

registerOnBlockchain
```
contract.registerBidder(address)
```

registerOnOffChainCode
```
url = "http://localhost/increment-number-of-bidder"
api.call(url)
```

Appendix 6: Enter as Seller

Inputs:
Using "Enter as SELLER" Button: showSellerModal

showSellerModal

```
    sellerModal.appear()
```

Appendix 7: Submit Asset Information

Inputs:
Using "Submit" Button: regiterAssetInfo
Using Form.Input: name;
Using Form.Input: description;

```
regiterAssetInfo
    contract.registerAuctionInfo(name, description)
    goToEnd()
```

Appendix 8: Nonce Generation

Inputs:
Using "Generate" Button: generateNonce
Using "I want to enter my own nonce" checkbox: unchecked

```
generateNonce
    nonce = ""
    characters =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
    for i in 0 to 30 {
        randomChar = getRandomChar(characters)
        nonce += randomChar
    }
    goToBid()
```

Appendix 9: Manually Entering Nonce

Inputs:
Using "Submit" Button: receiveNonce
Using "I want to enter my own nonce" checkbox: checked
Using Form.Input: nonce;

```
receiveNonce
    goToBid()
```

Appendix 10: Bid Phase

Inputs:
Using "Submit" Button: receiveBid
Using Form.Input: Bid;

receiveBid:
    hashOutput = hash256(Bid+nonce)
    submitHashToBlockchain(hashOutput)
    sendBidToOffChainCode(Bid)
    goToValidate()

submitHashToBlockchain(hashOutput)
    contract.submitHashedBid(hashOutput)

sendBidToOffChainCode(Bid)
    url = "http://localhost/submit-bid"
    api.call(url, bid)

Appendix 11: Validation Phase

Inputs:
Using "Validate" Button: validateClicked

validateClicked*:*
```
    max = -1000
    while max == -1000 {
      delayThreeSeconds()
       max = getMax()
    }
    if (max > bid) {
      neutral()
    }
    else if (max == bid) {
      winner()
    }
    else {
      dispute()
    }
```

getMax*:*
```
    url = "http://localhost/get-winner"
    winner = api.call(url)
    return winner
```

Appendix 12: Neutral (Confirmed) Case

confirmed:
```
    winner = 0
    while winner == 0 {
      delayThreeSeconds()
       winner = getWinner()
```

```
    }
    end()
```

```
getWinner()
    winners = contract.getAllWinners()
    return winners
```

Appendix 13: Winner Case

```
winner:
    winnerModal.appear()
```

Appendix 14: Winner Submission

Input:
Using "submit" button: submitClicked

```
submitClicked:
    contract.submitWinner(Bid, nonce)
    neutral()
```

Appendix 15: Dispute Request Case

```
winner:
    disputeModal.appear()
```

Appendix 16: Dispute Submission

Input:
Using "dispute" button: disputeClicked

```
disputeClicked:
    contract.dispute(Bid, nonce)
    neutral()
```

Appendix 17: Increment Number of Bidders (Off-chain Code)

Address: http://localhost/increment-number-of-bidders/

Type: GET

Input:
Using request: incrementNumberOfBidders

incrementNumberOfBidders in views.py

```
    controller.incrementNumberOfBidders()
    return response(202-accepted)
```

incrementNumberOfBidders in controller.py

```
    [...numberOfBidders] = readValuesFromModel()
    numberOfBidders += 1
    writeValuesIntoModel()
```

Appendix 18: Submit Bid (Off-chain Code)

Address: http://localhost/submit-bid/

Type: POST

Input:
Using request: submitBid
request;

submitBid in views.py

```
    bid = request["bid"]
    controller.submitBid(bid)
    return response(201-created)
```

submitBid in controller.py

```
    [...bids,numberOfBidders] = readValuesFromModel()
    bids.push(bid)
    numberOfSubmittedBids += 1
    writeValuesIntoModel()

    if numberOfBidders == numberOfSubmittedBids {
       controller.findWinner()
    }
```

findWinner in controller.py

```
    [...bids] = readValuesFromModel()
    winner = max(bids)
    writeValuesIntoModel()
```

Appendix 19: Get Winner (Off-chain Code)

Address: http://localhost/get-winner/

Type: GET

Input:
Using request: getWinner

getWinner in views.py

```
    winner = controller.getWinner()
    return response(data=winner, 200-ok)
```

getWinner in controller.py

```
    [...winner,numberOfBidders] = readValuesFromModel()
    [...numberOfAskedBidders] = readValuesFromModel()

    if winner == -1000 {
      return winner
    }
    numberOfAskedBidders += 1
    writeValuesIntoModel()

    if numberOfAskedBidders == numberOfBidders {
      controller.reset()
    }
```

reset in controller.py

```
    [bids, numberOfBidders, winner, numberOfSubmittedBids, numberOfAskedWinner] =
readValuesFromModel()
    winner = -1000
    bids = []
    numberOfBidders = 0
    numberOfSubmittedBids = 0
    numberOfAskedWinner = 0
    writeValuesIntoModel()
```