POLITECNICO DI TORINO

Master's Degree in ICT for Smart Societies

Master's Degree Thesis

FRONT-RUNNING ATTACKS IN A BLOCKCHAIN-BASED TRADING SYSTEM FOR ELECTRICITY IN PROSUMERS COMMUNITIES

Supervisors

Prof. MICHELA MEO

Prof. TAO HUANG

Prof. FERNANDO PEDONE (USI)

Candidate

MARK COMERRO

DECEMBER 2020

Abstract

Nowadays the continuous increase in the world energy consumption is bringing to the attention of everyone the problem of sustainability, since this demand still relies mainly on non-renewable energy sources. In this context, smart energy grids are emerging, as they allow to better exploit the distributed energy resources (DER), which are based instead on a renewable and thus more sustainable energy production. The thesis points out the front-running vulnerabilities and proposes prevention strategies to mitigate possible attacks in a blockchain-based trading system for electricity in prosumer communities.

An investigation of the blockchain front-running literature is presented in order to define the problem in the analysed project. A complete redesign of the smart contract is performed to implement front-running prevention strategies and to improve the performance in term of computational complexity. In addition, a new auction management approach to include electricity storage capabilities together with selling strategies is introduced to achieve a complete decentralised auction system and better exploit blockchain properties.

The thesis demonstrates that the front-running problem has to be evaluated on a case-by-case basis to propose custom solutions. In particular, the results prove that tailored front-running prevention strategies do not affect the functioning and the performance of the system.

Acknowledgements

I wish to express my sincere gratitude to my supervisors; Professor Michela Meo, who guided me during the entire master thesis project with her experience; Professor Tao Huang, who proposed me innovative ideas in prosumer communities context; and Professor Fernando Pedone from Università della Svizzera Italia of Lugano, who gave me a special support with his deep knowledge on blockchain topic by supervising me remotely cause of the COVID-19 pandemic. Without their persistent help, the goal of this project would not have been realized.

I wish to express my deepest gratitude also to my family, my father, Marco; my mother, Flavia; my brother, Manuel; and my aunt, Giovanna. They kept me going on and this work would not have been possible without their support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Smart grids framework

The meaning and the importance of prosumer communities cannot be fully understood without considering the global picture of the framework in which this concept was born. An interesting insight is given by the work of Espe et al [1], which reviews the recent literature around the concept of prosumer community based smart grids. They explain that nowadays the world energy demand is increasing at an unprecedented rate, and it is mostly supplied by non-renewable energy sources, which cause environmental damages. Therefore, an energy transition towards renewable sources is happening. It is here that *smart grids* come into play, deriving from the introduction of digital communication in the electricity grid, with the purpose of supporting this transition.

Smart grids allow the bidirectional exchange of energy and information on the grid, thus giving the chance for a better exploitation of the power produced by small-scale generators such as the distributed energy resources (DER), in which prosumers are involved.

According to [2], prosumers can be defined as households equipped with a decentralised production unit (DPU) for electricity, which is connected to the grid; this DPU can be a photovoltaic (PV) system or a micro wind turbine. Therefore, not only they consume electricity, but they are also able to produce it. The grid is exploited to reach the energy balance: since production and consumption energy profiles have different values along the day, when the electricity production exceeds the consumption, the energy surplus is exported to the grid; viceversa, when the energy need cannot be covered by the local production, the electricity is drawn from the grid.

In Europe, some legislation concerning the possibility to have prosumer communities can be found. In particular, the recast of the Renewable Energy Directive (Directive (EU) 2018/2001 [3]) provides a definition of *Renewable Energy Communities* (RECs) as legal entities (Art. 2.16), aimed at achieving environmental, economic or social benefits rather than financial profits. However, European countries have different legal frameworks for decentralised renewable energy (RE) production, as explored by the study of Campos et al. [4]. Among the nine EU countries under analysis there is for example Italy; their research work about the Italian legal framework shows that in Italy RE systems can be installed for self-generation, but households cannot sell the energy they produce among themselves.

In this heterogeneous scenario, there is space to investigate the creation and the management of prosumer communities, because they present many potential advantages for the society, in addition to the higher exploitation of renewable energy. For example, they could provide higher reliability of the electricity system, allow to reduce the investments for the infrastructure in the distribution network and bring higher social welfare through the economical competition.

1.2 Smart grids and blockchain

In the smart grids context, different technologies have been studied in order to provide hardware and platforms to ease the spread of distributed energy production from renewable energy sources.

This thesis takes into consideration a blockchain-based trading system for electricity in prosumers communities. The real innovation of the project is the utilisation of the blockchain technology, which offers a secure and decentralised way of recording the transactions payments. In particular, self-sustainable communities can be encouraged by the economic performances of a decentralised platform, where prosumers can decide for their own consumption and generation with selling and purchasing strategies.

Blockchain is an innovative and powerful technology that can support the development of numerous decentralised applications, but there is the possibility to encounter obstacles. The thesis takes into account the blockchain front-running problem with respect to the analysed system.

1.3 Blockchain and front-running

Front-running is the illegal practice of taking advantage of non-public market information regarding upcoming transactions and trades. The person who commits a front-running activity benefits from a privileged position along the transmission of information, in financial or non-financial systems.

Historically, traders might become aware of an upcoming large purchase by overhearing the negotiation between the broker and their client and then try to beat the broker's speed to buy first. Because of the large purchase, the front-runner takes advantage of the temporarily reduced supply of stock to make profit. Moreover, a broker can front-run their own clients by purchasing stocks for themselves before placing the offer of the client.

The first appearance of the practice of front-running was on the Chicago Board Options Exchange (CBOE), the world's first and largest organised stock options exchange. In 1977, the Chicago Board Options Exchange defined front-running as: "The practice of effecting an options transaction based upon non-public information regarding an impending block transaction in the underlying stock, in order to obtain a profit when the options market adjusts to the price at which the block trades [5]. In jurisdictions with established securities regulation, like CBOE, the front-running procedure is illegal and can be identified.

In a decentralised system, like blockchain, it is very difficult to identify and punish the front-runner. Blockchain technology is used to build decentralised applications (DApps), or smart contracts, in which each function call to the DApp consists in a transaction processed by a decentralised network. Moreover, blockchain introduces a new step in the process of finalising transactions called mining. Miners have the duty to select the transactions, put them into a valid block and incorporate it in the blockchain. Therefore, miners are in the best position to conduct a front-running attack, since they observe the transaction after it has been broadcast but before it is finalised. Malicious miners can attempt to have their own transaction confirmed before or instead of the observed one.

Front-runners can attack all kind of DApp, but it is not necessarily beneficial. It depends on the logic and mitigations that might be implemented in the DApp itself. Accordingly, DApps need to be studied individually or in categories. In this thesis, the front-running vulnerabilities of a blockchain-based trading system for electricity in prosumers communities are analysed and the possible solutions are provided. In particular, different categories of techniques to eliminate or mitigate front-running are discussed, including transaction sequencing, cryptographic techniques like

commit/reveal, and redesigning the functioning of the DApp to provide the same utility while removing time dependencies.

1.4 Goal of the thesis

This thesis fits into the context of DER and RECs and aims at investigating the front-running vulnerabilities of a blockchain-based trading system to allow electricity transactions in a community of energy prosumers and consumers. Moreover, the purpose is to completely redesign the system to implement front-running mitigation, add functionalities and enhance the overall performances.

The system is composed by different software actors that communicate to each others to offer a platform where the members of a community can register and exchange electricity. The use of the blockchain technology offers a secure and decentralised way of recording the transactions payments among prosumers. In particular, after the registration, users can sell and buy electricity, according to their electricity balance, trough an auction-based procedure.

The thesis aims to investigate the front-running vulnerabilities of the considered system. It is designed using an hybrid approach to integrate the blockchain advantages in a smart grids environment, which needs a strong relation with physical supports. Possible front-running mitigations are implemented, together with the introduction of the possibility to handle multiple contemporary actions for each user for different time-steps. Moreover, a redesign of the auction's life-cycle is performed in order to fully automatise the selection of the auction winner and to reduce the smart contract's function calls expenses.

Chapter 2

Front-running and Blockchain

2.1 Blockchain

2.1.1 Blockchain introduction

The blockchain, as its name suggests, is a chain of blocks which contains information. This technique has firstly been used by Satoshi Nakatomo in 2008 to create the Bitcoin cryptocurrency [6]. The blockchain is defined as a distributed ledger based on a peer-to-peer network in which everyone is allowed to participate. The most interesting property is security, in fact once the data are registered inside the blockchain, it is almost impossible to modify this information.

2.1.2 Cryptography

The blockchain systems extensively use cryptography techniques to ensure the ledgers' integrity. The integrity in this field consists in the ability to identify possible mishandling or compromise of the information stored in the blockchain. This property is crucial in public systems in which there is no predefined trust. Integrity is fundamental in the private blockchain too, since the authenticated nodes could also act in a wicked way [7].

The main function of the cryptography technique is to obfuscate a message and make it unintelligible to unintended readers. The sender uses an encryption algorithm, in conjunction with an encryption key, to encrypt the plain message before sending it. The receiver, once they receive the encrypted message, uses the correspondent decryption algorithm, combined with a decryption key, to decrypt

the message and make it readable. In this way, who does not own both the keys cannot read the data. There are two kinds of cryptography which differ by the type of the employed keys:

- the symmetric cryptography, in which the two keys coincide, so there is only one key used for both encrypting and decrypting the message;
- the asymmetric cryptography, a very complex technique in which the encryption key differs from the decryption key, but the two keys are generated in such a way that they can only work in pairs: the information encrypted by the first can only be decrypted by the second and vice-versa. Going into details, each user owns two keys: a public key to be spread as much as possible and a private key to be kept secret. To prove the authenticity of the message, it has to be encrypted with the private key of the author; in this way the receiver can decrypt the message with the sender's public key. On the other hand, to grant the confidentiality, the sender encrypts the message with the public key of the receiver, in this way only the addressee can decrypt the message with their private key, as is described in figure 2.1.

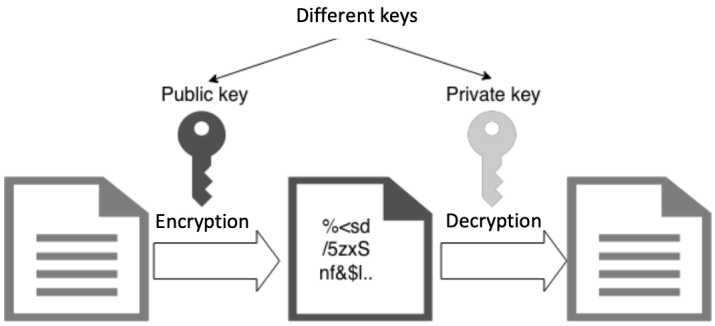

Figure 2.1: Asymmetric cryptography [8].

Since cryptography prevents unintended readers to have access to the message but does not grant the integrity, another step is needed with respect to the normal asymmetric one: the digest. It contains a data summary obtained through hash algorithms and it consists of a string which uniquely identifies that data. The hash algorithms have two main functions:

- from a string with arbitrary length they return a string with fixed length;
- they are irreversible functions, so from the result string it is impossible to retrieve the input string with arbitrary length.

The hash function used in the blockchain is the SHA256, which has the property of returning as a result an alphanumeric string with fixed length of 64 characters.

It is called SHA256 because it actually finds a binary number composed of 256 bits which results to be a string of 64 characters when it is transformed in hexadecimal format.

2.1.3 Block structure

In the blockchain, each block is uniquely identified by a hash generated by the SHA256 algorithm. The hash is calculated during the block creation and this string identifies the block itself and its content, including the hash of the previous block.

A block is a data structure composed by an header and a list of transactions. The header of the block consists of different fields: the previous block's hash, the *timestamp*, the *Merkle tree*, the *difficulty* and the *nonce* [9].

In the *Merkle tree* all the transactions contained in the block are present. This tree is obtained in the following way: transactions are paired (i.e., groups of two transactions each are formed) and the SHA256 algorithm is applied twice, in order to create a single hash representing each pair; this operation is applied recursively, until a unique hash for all the block, defined as the *Merkle Root*, is obtained.

It is possible to consider, as trivial example, a block with only four transactions: the *Merkle tree* structure is presented in figure 2.2. The *Merkle tree* is always binary and, if the transactions are odd, the last one is duplicated. In the end it does not matter how many transactions are present in the block, as they are always summarised in a 32-bytes hash.

2.1.4 Creation of a block

The peer-to-peer nodes are responsible for building a block and they are defined as *miners*; they validate the new transactions and register them on the distributed ledger. Mining is the mechanism that makes the blockchain secure and decentralised.

Miners compete in order to solve a computationally difficult mathematical problem based on the hash algorithm. The solution is called *Proof-Of-Work*. This is the demonstration that the miner has spent a large amount of time and computational and energetic resources (therefore economic ones, too). Once a block is solved, the contained transactions can be considered confirmed and so the involved coins can be spent.

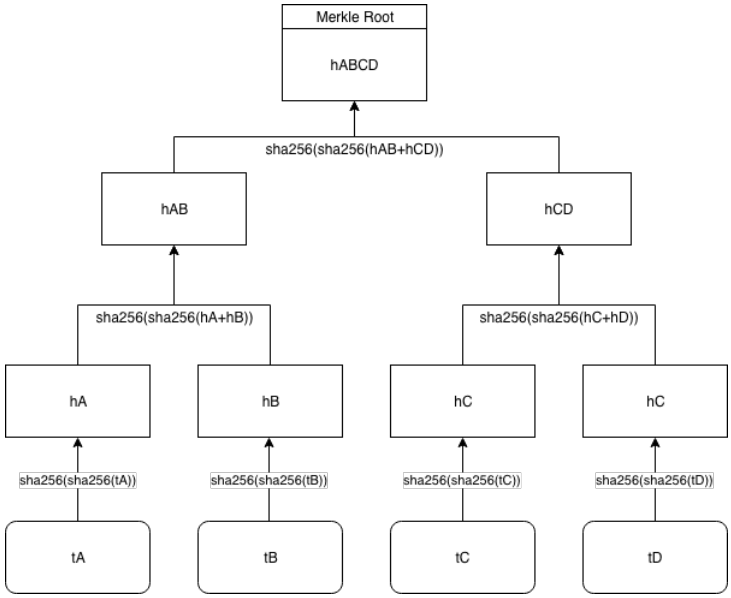

Figure 2.2: *Merkle tree* of a block containing four transactions [8].

When the miners solve the mathematical problem, they receive a prize that can consist of either new coins or the fees contained in the transactions.

- In the first case, every time a new block is created by a miner, the system gives them a reward which consists of a certain amount of crypto-currency created from scratch. The process of the creation of the new amount of coins is called *mining* because, like in a mine, the resources are destined to decrease over time. In Bitcoin, for example, this amount decreases approximately every four years.

- In the second case, the miner who wins the competition against the other nodes, and so the first to publish the solution, receives the fee of the transactions contained in the block. The creation of the new amount of crypto-currency is meant to decrease; on the other hand, fees are destined to increase over time.

While a miner is waiting to find the *Proof-Of-Work* of the current block, they are also listening to the network for new pending transactions. These new transactions are added to the *memory pool*, or *transaction pool*, where they wait to be included in a new block and validated.

When a miner is informed that the current block has a valid *Proof-Of-Work*, they begin to build a *candidate block* by assembling the pending transactions in the *transaction pool*, selecting the ones with greater priority. The block is defined

as a candidate because it does not own a valid *Proof-Of-Work* yet.

The first thing that a miner does is to create a *coinbase transaction* which contains the reward of the mined block. This transaction is different from all the others because the coins are created from scratch. The second step is to create the *header*, and in order to do that a miner has to compute the previous block's hash. Then, the *Merkle Root* is obtained as the summary of the transaction contained in the block and so by the *Merkle tree*. The nodes also add the *timestamp* and a field named *difficulty*, which represents the complexity of the mathematical problem that has to be solved in order to insert the *candidate block* into the blockchain. This parameter is given by the system and it varies over the time, it is designed to keep approximately constant the average time needed to add a block to the chain. The last field is the *nonce*, which consists of a string of data that changes randomly over time until the hash of current building block begins with the number of zeros contained in the *difficulty* parameters. When the right number of zeros is found, the block can be successfully inserted in the blockchain, as represented in figure 2.3 [9].

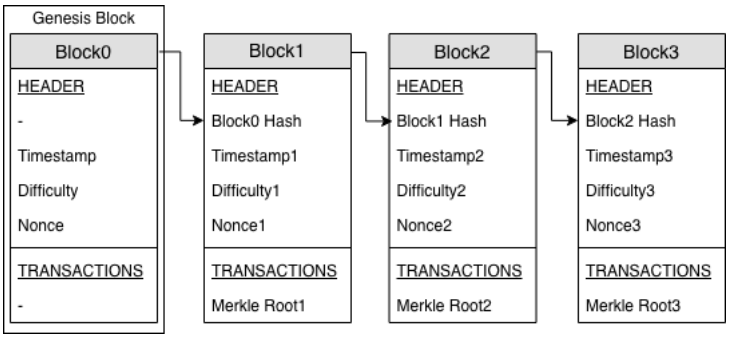

Figure 2.3: New block addition to the blockchain [8].

2.2 Ethereum and DApp

2.2.1 Ethereum introduction

Blockchain technology is constant evolving and one of the latest developments is the creation of the smart contracts. Smart contracts are simple programs saved on the blockchain and they can be used to automatically exchange crypto-currencies under predefined constraints.

The most widely used blockchain that supports smart contract is Ethereum. Ethereum, created in 2013 by Vitalik Buterin, is designed to enable the smart

contract implementation in a specified programming language called Solidity, which uses a JavaScript-like syntax. The Ethereum blockchain is very close to the previously described one, the only difference is the possibility to create operations with any level of complexity thanks to the Patricia-Merkle tree. In particular, Patricia-Merkle tree consists of three different trees for three kind of objects [10]:

- *Transaction tree* which is equivalent to the above described Merkle tree;
- *Receipts tree* containing data related to the result of each transaction;
- *State tree* representing the current state of the blockchain .

Each user in the blockchain is uniquely identified by the address of the respective wallet. The wallet allow the user to deploy smart contracts and store crypto-currencies, while the address, used to send and receive transactions, consists in a unique series of letters and numbers.

2.2.2 Gas

In the Ethereum blockchain, the unit of measurement of operation's computational complexity is called "Gas". In particular, a Gas quantity is associated to each transaction together with the gasPrice, which corresponds to the amount of Ether (Ethereum crypto-currency) that the sender will pay to the miner for the performed computational effort. The Gas functions are mainly two:

- guarantee a prize for the miners which execute the code and maintain secure the blockchain;
- ensure that the transaction's execution will not overcome the estimated time limiting the founds loss in case of encountered errors.

It is important to notice that Gas is a unit of measurement of computational complexity of a transaction and not a unit of measurement of the code length. The code written in Solidity can be arbitrarily complex, short algorithms can generate large amount of calculations, while longer algorithms can generate less computational effort [10].

2.2.3 Smart contract

The key elements of Ethereum are the smart contracts, which can contain any kind of algorithm that will be executed by each node during the creation of a new block. Moreover, smart contracts are able to receive and send transactions between each others.

Each smart contract functioning is uniquely identified by an address on the blockchain, which is assigned by a node special transaction during the contract creation. Subsequently to the initial transaction, the smart contract will become permanently part of the blockchain and its address will never be modified. In order to call a method on the smart contract, a node has to send a transaction to the address of the smart contract specifying the method and the input parameters. The called method will be executed by the smart contract during the creation of the new block. Each method can return values or store data on the blockchain state [11].

2.2.4 DApps

The goal of Ethereum is to revolutionise Internet functioning, for the first time is is possible to develop information systems without the need of third parties. The classical network is replaced with a decentralised network able to support anyone utilisation of decentralised web applications (DApp). DApps are based on the smart contracts, which are impossible to be modified or manipulated and autonomously work without intermediate entities as represented in figure 2.4.

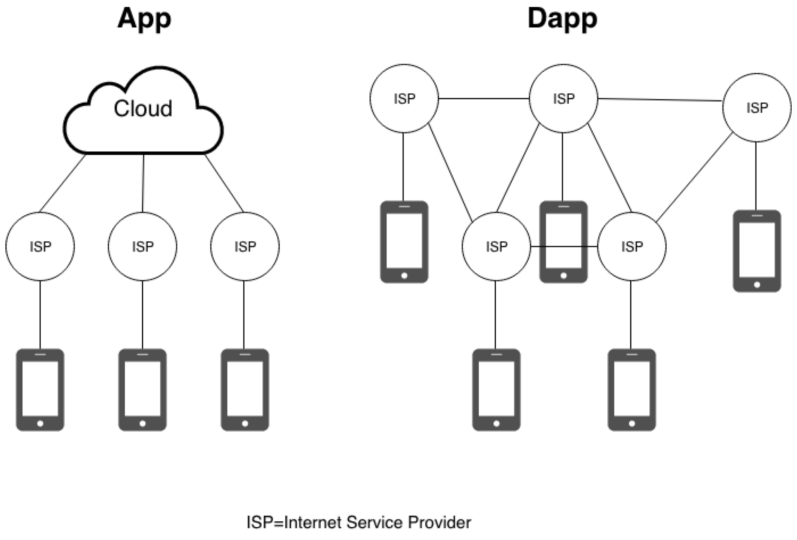

Figure 2.4: Logic scheme of a DApp [8].

2.3 Advantages and disadvantages

The blockchain offers a lot of opportunities and applications in a continuously increasing number of fields, but it is not always the best choice. For this reason,

it is important to evaluate the advantages and the disadvantages, which can be summarised as follows [11].

- Advantages:
  - implementation of a distributed database maintained by nodes in which each of them has access to the data and sees every transaction. This kind of information storage prevents the loss of data in case of unexpected events;
  - thanks to the cryptographic algorithms, the trust between the parties is granted without intermediaries;
  - it can potentially become a global database in which everyone can write and read;
  - the transparency is granted. Everyone can read not only the final state of the transactions, but also the complete states' history;
  - immutability, since data cannot be modified or deleted;
  - decentralization, as the blockchain can exist without a central authority and it cannot be manipulated, censured or interrupted.

- Disadvantages:
  - it is characterised by a high energy consumption for each transaction. In Bitcoin, for example, a transaction can cost 6 dollars considering the energy consumed by each node;
  - the mining operation needs very expensive hardware and the majority of the computational power is wasted;
  - the local copy of the entire blockchain on each node needs a large amount of storage capability;
  - the addition of new information is slow. In Bitcoin, for example, the creation of a block can take from 10 to 60 minutes. In Ethereum it takes 15 seconds;
  - immutability and transparency can damage the privacy and the reputation of the users;
  - smart contracts cannot refer to external libraries, therefore the data needed by the smart contract has to be inserted in advance in the blockchain;
  - smart contracts can contain errors in the code, usually called *bugs*.

Taking into consideration these advantages and disadvantages, it can be understood why the blockchain technology is suitable for the purposes of this project. In particular, since the framework of distributed energy production is considered, also the data regarding the energy transactions should be distributed, without the need of a trusted third party, and this can be achieved through the use of the blockchain. Secondly, the possibility of having a programmable blockchain allows to define smart contracts to regulate these energy transactions. In addition, the blockchain also offers a secure environment for the bilateral negotiations among prosumers.

2.4 Front-running in Blockchain

2.4.1 Taxonomy of front-running attacks

Font-running attacks on blockchain can be often reduced to one of a few basic templates. The common circumstances could be that Alice is trying to invoke a function on a contract that is in a particular state, and Mallory will try to invoke her own function call on the same contract, in the same state, before Alice. Focusing on what the malicious subject is trying to accomplish, it is possible to identify three main cases [12]:

- The first case is the *displacement attack*, in which it is not important to the adversary for Alice's function call to run after Mallory runs her function. Alice's transaction can have no meaningful effect or be orphaned. For example, Alice trying to register a domain name and Mallory register it first or, in an auction-based system, Alice trying to submit a bid in an auction and Mallory submit it first.

- In the second type of attack, the *insertion attack*, Mallory runs her function, the state of the contract change and the adversary needs that Alice's original function run on the modified state. In order to better explain the situation, it is possible to imagine that if Alice places a purchase order on a blockchain asset at a higher price than the best offer, Mallory will insert two transactions: she will purchase at the best offer price and then offer the same asset for sale at Alice's slightly higher purchase price. If Alice's transaction is then run after, Mallory will profit the price difference without even hold the asset.

- In the last case, the *suppression attack*, after Mallory runs her function, she tries to delay Alice from running her function. After the delay, she is indifferent to whether Alice's function runs or not. Alice's transaction can have no meaningful effect or be orphaned.

Each of these attacks has two variants, *asymmetric* and *bulk*. The attack is defined asymmetric if Alice and Mallory are performing different operations. Instead, the attack is called bulk if Mallory is trying to run a large set of functions for a limited set of assets offered on a blockchain.

2.4.2 Cases of front-running in DApps

In order to better understand the effect that a front-running vulnerability can have, it is important to give some examples from the most popular DApps. The authors of "Transparent Dishonesty: Front-running Attacks on Blockchain." [12] categorised the top 25 DApps from `DAppradar.com` in September 2018 into four principal use cases.

The fist category of DApp, *Markets and Exchanges*, are financial exchanges for trading ether and Ethereum-based tokens. Two different situation are analysed under the front-running vulnerability aspects. The fist considered DApp is a partially centralised exchange called EtherDelta. As in traditional financial markets, one method to manipulate the spot price of an asset, called "taker's griefing attack", consists in flooding the market with orders and cancel them when there are filling orders. In EtherDelta, to prevent taker's griefing attacks, the user needs to send an Ethereum transaction to cancel each of his orders. In this particular case, a front-runner can send a fill order transaction with higher gasPrice to get in front

of the cancellation order and take the order before it is canceled. The described attack is known as "cancellation grief" and follows the asymmetric displacement template.
Another example of financial exchange DApps is Bancor. Bancor provides continuous liquidity for digital assets without relying on brokers to match buyers with sellers. In Ethereum blockchain, as described in section 2.2, when transactions are broadcast to the network, they wait to be mined in a pending transaction pool known as mempool. Since Bancor is based on Ethereum, the DApp transactions are visible to everyone for some moments before being inserted in a block, leaving Bancor vulnerable to front-running. Attackers can take advantage of the pending time and gain profits by buying before the order and fill the original order with slightly higher price.

The second use case of DApp is *crypto-collectables*. In particular, Cryptokitties is the most popular DApp in the crypto-collectables category and third most active overall. The DApp is designed around kitties, each of them is a cartoon kitten with a set of unique features that define the rarity of the kitten itself. Kitties can be bought and sold through auctions on the Ethereum blockchain. Moreover, kitties can breed and give birth. When the two parents breed, the Cryptokitties smart contract define from which future block the pregnancy can be completed. From that specific moment, anyone can complete the pregnancy by calling `giveBirth()` method and collect a reward in cryptocurrency. The attacker can perform a displacement attack to front-run the `giveBirth()` function call in order to obtain financial profit.

The third identified category of DApp is *Gambling Services*. Fomo3D is the most popular game on Ethereum in the sample taken in consideration by [12]. The goal of the game is to be the last user to have purchased a ticket when a timer goes to zero, anyone can buy a ticket and each purchase increases the timer by 30 seconds. The popular opinion was that the game would never end but on August 22, 2018, the first round of the game ended. The winner gained 10,469 Ether equivalent to 2.1M USD at the time. Deep blockchain analyses indicate that a very peculiar winning strategy was implemented to displace any new ticket purchases that would reset the counter. In particular, the winner seems to have started by deploying a lot of DApps unrelated to the game that consume high amount gas. Then, the winner placed multiple high gasPrice transactions to their DApps after they both 1 ticket at 3 minutes before the timer of the game reached zero. These transactions congested the network and, since they had very high gasPrice, miners prioritised them ahead of any new ticket purchases in Fomo3D. This attack is classified as a suppression attack in the taxonomy presented in 2.4.1. The key difference between the suppression attack and the displacement attack is that the front-runner, as in the Fomo3D case, does not care at all about the execution of their transactions.

The only goal of the attacker transactions is to delay all the others transaction of at least 3 minutes.

The final use case is *Name Services*, which decentralise the web domain registration. Ethereum Name Service (ENS) is the most active naming service on Ethereum. New .eth domain names are sold in a auction-based system, where the `startAuctionsAndBid()` method leaks the hash of the domain and the initial bid amount in the auction. In particular, users can bid for 3 days before the 2-day reveal phase begins, in which all bidders (winners and losers) must send a transaction to reveal their bids for a specific domain or sacrifice their bid amount. It is important to notice that if two user bid the same amount, the first to reveal wins it. The attacker, using the leaked information, can steal the domain name and win the auction with the same price of the original bidder by revealing it first.

2.4.3 Front-running prevention strategies

In the same way as front-running attacks strongly depends on the DApp implementation also the front-running mitigations must be investigated for each specific DApp. In the different prevention strategies, it is possible to highlight common patterns and classify them in three main categories: *Transaction Sequencing*, *Confidentiality* and *Design Practices* [12].

The first category, *Transaction Sequencing*, is strictly related to the blockchain on which the DApp is designed. The idea is to give a sequence to the transactions to force an order during the mining process. Transaction priority is something that depends on the implementation of the blockchain itself. For example, Ethereum pending transactions are stored in the pools and miners draw from them when forming blocks. As the term "pool" implies, there is no intrinsic order to how transactions are drawn and miners are free to sequence them arbitrarily. Miners can sequence transaction based on their personal preference. In most of the case, miners, to maximize their profit, order the transactions with respect to their gasPrice.
Force a transaction sequencing in a distributed network is generally not possible. Because pending transactions can reach different nodes in a different order. The solution could be to assign sequential number to transactions relying on a third trusted party, but it is contrary to blockchain's distributed trust. Although, it is possible in the smart contract implementation, to design functions which accept transactions that specify the current state of the contract as the only state to execute on. In this type of implementation, transactions themselves enforce an order over an existing blockchain.

In the second category, *Confidentiality*, the mitigations are focused on techniques to increase confidentiality and anonymity in DApps. In blockchains, all transactions are public and, by extending these confidentiality protections, the attacker will not know the transaction they are front-running. In particular, a DApp interaction includes the following components: the identity of the sender, the address of the contract the function is being invoked on, the name of the function being invoked, the code of the DApp, the current state of the DApp and the parameters supplied to the function. The applicability of confidentiality techniques must be evaluated on a case-by-case basis in order to limit the public available information.
The *commit/reveal* approach, together with sequencing, can be integrated in smart contract to protect the function call and the respective parameters. Once the DApp receive the function call and the sequence is established, the confidentiality is lifted and the function can only be executed in the order it was enqueued. One way to implement this strategy is to evaluate and send the cryptographic hash of the parameter with a random nonce to the smart contract. Then, once the sequencing is performed, reveal the original parameter and nonce to the smart contract which can verify the correctness of the previous commitment. It is important to notice that it is a two-way communication approach that results in an increased gas to pay, because of the two transactions needed. Moreover, aborting after the first round could be an issue for the smart contract execution.

The last approach to mitigate front-running is called *Design Practices* and assumes that front-running is unpreventable, therefore the only way to prevent it is a redesign of DApp's functionalities to remove any benefit from it. For example, a method to discourage attackers can be to reduce time-dependency from the DApp.

It is important to notice, that each DApp and each smart contract must be studied individually, in order to highlight specific font-running vulnerabilities and possible mitigations. The purpose of the thesis is to analyse the case of a blockchain-based trading system for electricity in prosumers communities and, if needed, implement front-running protection strategies without affecting the predefined requirements of the system.

Chapter 3

Blockchain-based trading system for electricity in prosumers communities

3.1 System composition

3.1.1 System overview

In figure 3.1 it is possible to see how the system is organised in terms of actors and their communication.
The core of the system is the web application, which behaves as a bonding element between the prosumer community and the rest of the software actors, as described in section 3.1.5. It is important to underline that each member of the community interacts with the web platform in two ways:

- human-machine interaction. The user utilises the web application from their computer to make energy and money-related decisions;
- smart meter-platform interaction. The user's smart meter periodically sends their energy balance to the system.

The web application also handles the majority of the data present in the system. Personal information of the users are stored in a MongoDB collection, while the smart contract is in charge of gathering data related to the decentralised monetary transactions, as written in section 3.1.3. The system also needs to use an already existing electricity grid, that is simulated using MatPower, as described in section 3.1.4. To this purpose, a Python web server is implemented to act as an interface between the just mentioned MatPower simulation (implemented using MATLAB

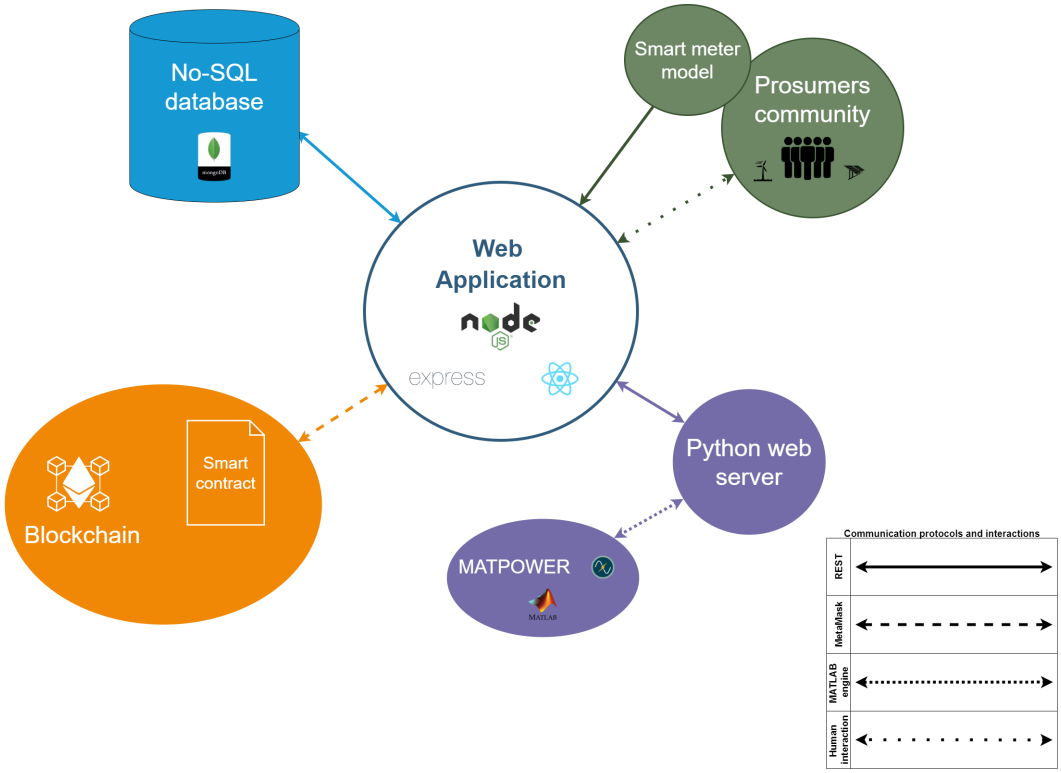

Figure 3.1: System overview: actors and communications.

language) and the system itself.

Prosumers community identify a group of households or micro-industrial community. From the electricity grid point of view, each user is identified by their smart meter which is responsible to records information such as consumption and production of electric energy. Smart meters communicate the information to the consumer for greater clarity of consumption behavior, and electricity suppliers for system monitoring and customer billing. In the implemented system, the smart meter corresponds to a Python script implementing a RESTful web service and a basic energy balance evaluation.

3.1.2 Key technologies

The key technologies that have been used in the project and the corresponding descriptions can be summarised in the following list.

- *Blockchain* technology is used to record monetary transactions among the users, and a *smart contract* developed on top of it serves as a regulator, as

will be explained in section 3.1.3. In particular:

  - *Ethereum* blockchain has been chosen among the different blockchain environments, for the reasons described in section 2.2.
  - *MetaMask* is a browser extension for Google Chrome, which can be found at [13]. It provides each user with a wallet address, corresponding to an Ethereum wallet, which contains the user's Ethers, i.e. Ethereum's cryptocurrency. MetaMask manages wallets automatically; users only have to create a password to access their account, and they can share the public key that they are given, if they want to receive Ethers from someone. Moreover, from Metamask the user can decide also to connect to the main Ethereum blockchain or to other networks.
  - During the development phase of the smart contract, it can be useful to work on a local blockchain, instead of the real one, in order to run tests without losing real Ethers. *Ganache* is a tool included in Truffle Suite [14] and it is used for this purpose. It provides a local blockchain with up to 100 fake accounts, which can be easily imported in Metamask, so that they can interact with the web application of this project.
  - *Remix Integrated Development Environment (IDE)* is an online tool where it is possible to build Ethereum smart contracts with Solidity language. It has been exploited to deploy the smart contract on Ganache virtual blockchain, test all its functions and debug the corresponding transactions, to check that the execution was performing as expected.

- The power flow simulation is used to check the feasibility of the electricity transactions derived by the bilateral negotiations among users. It comprises three main components:
  - *MATLAB Engine API for Python*, which provides a Python package named *matlab* that allows to call MATLAB functions directly from Python scripts [15].
  - *MatPower*, which is a MATLAB package for solving power flow and optimal power flow problems [16].
  - *CherryPy*, which is an object-oriented web framework, that allows to implement HTTP servers in Python [17].
- The user interface chosen for the system is a web application implemented following the *MERN* stack design, which consists of four main components:
  - *MongoDB* is a document-oriented database which stores data in JSON documents to make them flexible [18].

- *Express* is a *Node.js* framework which provides a robust set of features and HTTP methods for the web application [19].
- *React* is a JavaScript library to build components-based user interfaces [20].
- *Node* is the event-driven JavaScript runtime environment on which the web application is built [21].

3.1.3 Smart contract

The system presented in this thesis allows the users to exchange energy and complete the corresponding payments using cryptocurrency, directly on the platform. In order to regulate the money transactions, a smart contract is needed. The smart contract that is developed on Ethereum blockchain make use of Solidity as their programming language. This allows to directly manage payments and accounts, without using third-party libraries [22]. A smart contract is compiled and run on the Ethereum Virtual Machine (EVM), namely on top of the blockchain.

The choice of Ethereum as the blockchain where to implement the smart contract for this project has not been dictated only by its suitability for decentralised application development. One important aspect is also that Ethereum is the most popular blockchain, so it has more nodes participating to the network and it is more difficult for a group of malicious users to gain enough CPU power to attack the blockchain. Moreover, Ethereum results more appealing than the other blockchains because it has the lowest energy consumption due to the mining process [23], which is an important feature for a project of this kind, inserted in a sustainability perspective. In addition, the speed at which transactions are recorded on Ethereum blockchain is higher, since 15 seconds are required to create a block, while from 10 to 60 minutes would be needed for Bitcoin blockchain [11].

The smart contract (shown in appendix A) that has been implemented for this project has the following methods:

- `newUser` function is called when a user registers on the web application, the smart contract also registers them and keeps memory of the association of the user's smart meter ID with the user's wallet address;

- `createAuction` method is implemented to handle the situation of an already registered user who wants to sell electricity. In particular, to create a new auction on the smart contract and make it available on the market, the user has to specify the selling strategy and the base price;

- `placeBid` can be used by other users to place bids in the specified auction by sending the offer to the smart contract itself. If transaction value is be grater than the current best one, the money of the bid is transferred to the smart contract and the previous owner of the best offer is refunded by the smart contract itself. This makes sure that the bidders actually pay the money that they offer;

- `closeAuction` is called by the owner of the auction (the seller) to close the specified auction, the method establishes the winner based on the strategy defined during the creation of the auction and the sellers get their money from the smart contract.

3.1.4 MatPower

The energy transactions that take place in the prosumers community are the outcome of the bilateral negotiations among the users. In order to manage the power flows inside the system and check the feasibility of the electricity exchanges among prosumers, MatPower has been employed as simulation tool. MatPower is an open-source power system simulation package of MATLAB for solving AC and DC power flow and optimal power flow (OPF) problems [24].

The input file of a MatPower simulation is called *MatPower case* and consists of a MATLAB M-file or MAT-file which returns a single MATLAB struct [16]. In particular, a case is used to model a real power system, where each node of the grid corresponds to a so called *bus*. In this thesis, each bus has been assigned to one user of the community. Moreover, MatPower cases are identified by a number, that defines the number of buses in the considered distribution network. Each case contains a set of four matrices:

- *bus data*, in which each bus of the system is defined as load, generator or slack bus[1] by a specific type value (respectively 1,2 and 3). Moreover, this matrix also specifies the real and reactive power demand of the buses;

- *generator data*, that specifies for each generator of the network the amount of real and reactive power produced and the minimum and maximum values of production;

- *branch data*, in which both physical characteristics of the lines, such as reactance or resistance, and the branch flow limits are specified;

[1]The slack bus is used to guarantee the energy balance in the distribution network, by compensating the system losses.

- *generator cost data*, that defines the costs for active and reactive power produced by the generators of the system.

In this work, a predefined case has been considered (predefined MatPower case: *case9*) and, in order to adapt its structure to the features and purpose of this power system simulation, some modifications have been made. In detail, the first part of this customisation process consisted in setting to zero all the loads and the generators present in the original case. Moreover, the slack bus that is used in the simulation to model the Italian electricity provider has been set as a generator with almost infinite power. In addition, flow constraints on the distribution lines have been adapted to the amount of electricity exchanged among the system's users.

As already presented in section 3.1.2, the power flow simulation has been performed using MatPower, the MATLAB engine API for Python and the CherryPy library. In the second part of this customisation process, a RESTful web server has been implemented using CherryPy in order to put MatPower in communication with the other components of the system, such as the web application and the Python simulation. Moreover, the MATLAB Engine API has been employed to run MatPower functions directly from Python. Then, the chosen case has been customised using data coming from the external environment.

In MatPower the DC power flow is calculated using Newton-Raphson method through the built-in function *runDCpf*. Once performed the simulation, the results are packaged into a MATLAB struct and pretty-printed to the screen. Figure 3.2 shows the results of a MatPower DC power flow simulation and the system summary, which specifies the number of generators (prosumers), the number of loads (consumers) and the total power generated in the system. The bus data, instead, displays in a more intuitive way the amount of electricity sold or purchased by each bus (or user) of the community.

Then, in order to check the feasibility of the power transactions derived from the bilateral negotiations among users, it is necessary to verify that the branch flow constraints are not violated. In particular, the amount of power flowing on each electrical line of the system must not exceed the physical limits of the line itself, thus ensuring a proper operation of the entire distribution network. To this purpose, at the end of each DC power flow simulation a feasibility check on each line is performed and a boolean result (True if feasible, False if not) is returned to the external environment using REST. It should also be mentioned that a MatPower simulation is performed when a user place a bid, and the transaction is confirmed only if the feasibility check is positive.

```
================================================================================
|     System Summary                                                           |
================================================================================

How many?                How much?              P (W)            Q (VAr)
---------------------    --------------------   -------------    -----------------
Buses              9     Total Gen Capacity     650.0            0.0 to 0.0
Generators         4     On-line Capacity       650.0            0.0 to 0.0
Committed Gens     4     Generation (actual)    230.0                0.0
Loads              3     Load                   230.0                0.0
  Fixed            3       Fixed                230.0                0.0
  Dispatchable     0       Dispatchable          -0.0 of -0.0       -0.0
Shunts             0     Shunt (inj)            -0.0                0.0
Branches           9     Losses (I^2 * Z)        0.00               0.00
Transformers       0     Branch Charging (inj)    -                  0.0
Inter-ties         0     Total Inter-tie Flow    0.0                 0.0
Areas              1

                           Minimum                      Maximum
                  -------------------------  ---------------------------
Voltage Magnitude   1.000 p.u. @ bus 1         1.000 p.u. @ bus 1
Voltage Angle      -0.00 deg   @ bus 3         0.00 deg   @ bus 7
```

(a) Power system summary.

```
================================================================================
|     Bus Data                                                                 |
================================================================================
 Bus      Voltage          Generation             Load
  #   Mag(pu) Ang(deg)   P (W)     Q (VAr)    P (W)     Q (VAr)
-----  ------- --------  --------  --------  --------  --------
    1  1.000    0.000    -0.00       0.00       -         -
    2  1.000    0.000       -          -       50.00      0.00
    3  1.000   -0.000       -          -      150.00      0.00
    4  1.000    0.000       -          -       30.00      0.00
    5  1.000    0.000       -          -         -         -
    6  1.000    0.000     50.00      0.00        -         -
    7  1.000    0.000    150.00      0.00        -         -
    8  1.000    0.000     30.00      0.00        -         -
    9  1.000    0.000       -          -         -         -
                         --------  --------  --------  --------
               Total:    230.00      0.00    230.00      0.00
```

(b) Bus data.

Figure 3.2: DC Power flow results.

3.1.5 Web Application

The interaction between the user and the system is implemented through a web application that follows the *MERN* stack design. Data are mainly stored in a MongoDB database, but since the real innovation of this project is the utilisation of the blockchain, there is the need of an interaction between the web application and the smart contract described in section 3.1.3. This kind of communication is enabled by MetaMask, which is a browser extension that allows the access to the Ethereum enabled distributed applications by injecting the Ethereum web3 API in every website implemented in a JavaScript context. In this project MetaMask is used to manage the user wallet and their transactions, in fact, to let the web application work properly, the user has to to login in this particular extension.

In order to match the project purpose there is the need to store some personal information related to power grid. In this way the web application can directly communicate with the smart meter to receive the power balance. Moreover, part of these data are transmitted to the MatPower server during the place bid procedure in order to validate the offer. This implies that the web application cannot be a fully decentralised application (DApp). A central trusted authority for the grid point of view is needed to store these sensible data in a secure database. The main advantage of this hybrid solution is to grant both the economical transaction decentralization and the necessary interaction with the grid.

Before starting to use the web application, the user has to complete the registration phase, which consists of the two steps showed in figure 3.3:

1. fill in the registration form, which is presented in two slightly different versions (one for prosumers and one for simple consumers). In this project, the typology of the user is identified by the last number of their smart meter ID, which

is 0 for consumers and 1 for prosumers. When a user attempts to register themselves as a prosumer, the energy source of their production is also needed (wind or sun);

2. confirm the registration transaction to the smart contract trough the MetaMask pop-up.

The personal information is stored in a MongoDB database using the smart meter ID as a unique identifier. Thanks to this auxiliary database, in the smart contract only the wallet address and smart meter ID are collected. Since the blockchain is public, in this way the privacy requirement can be ensured.

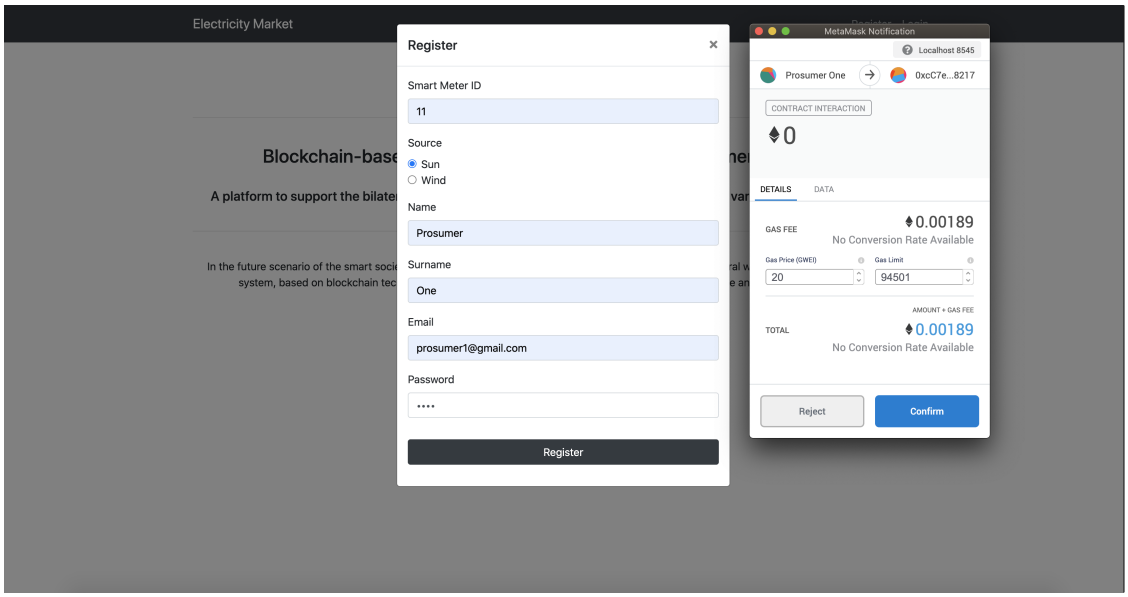

Figure 3.3: Prosumer registration into the web application.

After that the transaction has been confirmed, the user can access the functionality and the data available on the web application. The front-end showed in figure 3.4 consists of the app bar at the top (which contains user information) and of a layout composed by three tabs: *Market*, *Offers* and *Transactions*. Focusing on the *Market*, it is possible to identify the representation of the market prices history grouped by source and the *Auctions* section containing the list of the current available energy auctions. The *Offers* tab contains all the received and placed offers, while the *Transactions* tab shows the personal transactions history.

The interaction between the user and the web application can be summarised in three phases corresponding to the auction life-cycle:

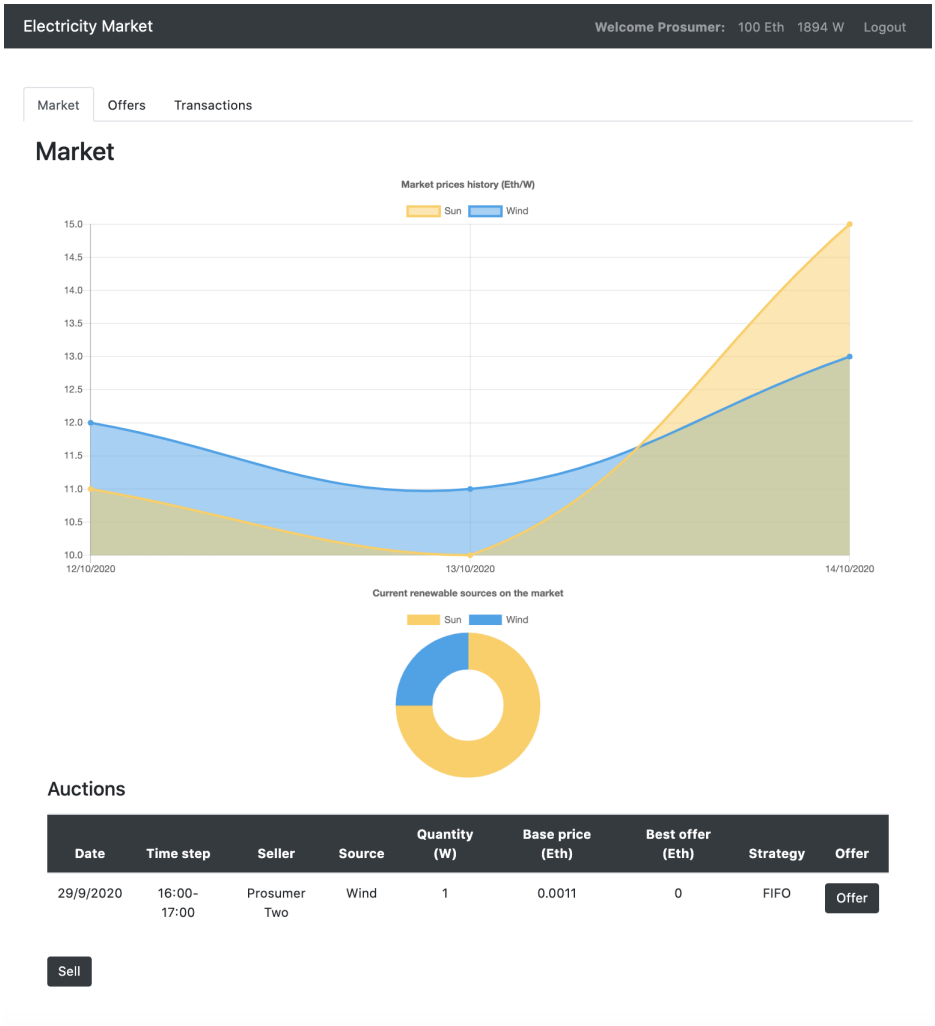

Figure 3.4: Prosumer home page.

1. *Selling phase*. The first phase begin with the *sell* button in the *Market* tab. During this phase a prosumer can put on the market their available energy defining the strategy, the quantity of electricity and the correspondent price. After the submission, the user has also to confirm the transaction to MetaMask in order to register the auction also in the smart contract. Once the procedure is successfully completed, the auction is on the market and it is able to collect the bids that arrive during the buying phase.

2. *Buying phase*. Once the auction is on the market, the user can place their bid by clicking the *offer* button of the preferred active auction and defining amount of cryptocurrency they want to offer. In this moment, a call to the

grid feasibility check is performed. If the transaction is feasible from the grid point of view and the MetaMask transaction is confirmed, the offer is registered both in the database and in the smart contract. Moreover, the corresponding amount of eth is sent to the smart contract itself. In this way, the system can prevent double spending and it can grant the payments.

3. *Closing phase*. Finally, the seller who received at least a bid from a buyer can terminate the auction clicking on the *Close* button of the respective auction in the *Offer* tab. The smart contract select the best offer under the predefined strategy criteria and transfers the crypto-currency to the seller. The auction is now closed and it is no more available on the market.

3.2 System improvements

3.2.1 Auction management

One of the main improvement to the system that the thesis proposes is the enhanced auction management. The original system is based on a time-steps division of the time. At the end of each time-step each user must have an electricity balance equal to zero. In particular, prosumers have to sell all their energy and consumers have to buy as much energy as possible from the market. This process is iteratively performed during each time-step and the balance estimation is computed referring to the immediate next time-step. Users can only buy or sell for the following time-step and only one auction for user is available. A simile approach does not take into consideration the personal needs and preferences of the user. Moreover, the integration of a electricity storage system would not be admitted because of the system constraints.

The presence of a battery is crucial for the energy management, the user can take decisions on their own electricity and sell it to the community when they prefer. The storage capability enables more complex market dynamics, for example prosumers can maximise profits by buying in surplus when the price of electricity is low, store the energy and sell it during an electricity shortage at much higher price.

The thesis project introduce a more sophisticated auction management approach. A completely redesign of the auction process that span from the Python server to the smart contract trough the web application is performed. Users can sell or buy electricity for any future time-step, regardless the time-step they are in. Moreover, a user can manage multiple contemporary auctions, each of them with its own price, quantity, time-step and strategy (the auction's strategy is explained in section 3.2.2). This new auction management implementation enables a wider range of

opportunities for the user, making the system compatible with any kind of electricity storage equipment. Prosumers can sell their electricity during the time-step they prefer or they can buy, store and then resell the same electricity to make profit.

For what concern the electricity grid validation in MatPower, the Python server can now keep track and update all the future transactions to always keep synchronised the market and the smart grid model loads. Moreover, the entire system enhancement has been made with the goal of do not affect the user interface, in order to maintain the continuity in term of user experience.

3.2.2 Selling strategies

The second improvement that worth to be highlighted is the introduction of the selling strategies. This addition enables the complete utilisation of the blockchain's decentralised transparency. In the original implementation of the system, the winner selection is made by the seller outside the blockchain and criteria are not public available. This selling scheme is subject to complaints about possible unfair decisions.

In the updated system proposed by the thesis, the auction winner selection is completely shifted to the smart contract. During the selling phase, prosumers select one of the available selling strategy, the selection is communicated to the smart contract, then it is utilised to identify the best offer under the defined criteria. The system is designed to accept any kind of selling strategy and it is already proposed with two basic strategies:

- the first one is *FIFO* which stands for "First In First Out". As the name suggests, the first received bid is the one that wins the auction and the cryptocurrency transaction is immediately performed by the smart contract.
- the second implemented strategy is called *Max price* and if selected, the smart contract choose the winner as the bid corresponding to the maximum amount of ether. The smart contract accepts new bids only if their are better than the current best one. If a new better bid is received, the smart contract refund the owner of the previous best offer and collect the new one. When the seller close the auction, the smart contract complete the transaction and send the money to the seller.

Once the auction is on the market, the seller has no more power of decision on the winner selection, the criteria are public available and the transparency is ensured.

During the selling phase, the strategy selection enables also the update of the future electricity transaction in the grid model in MatPower. The Python

server receives the new best bid and immediately update the state of the grid model in order to perform the feasibility check on the lines and loads. If the transaction is positively assessed, the state of the grid model is updated and the bid is communicated to the smart contract for the previously described procedures.

3.2.3 Smart contract efficiency

The original smart contract implementation is designed to collect all the bids from the market in potentially big data structures. In particular, each auction is stored in a mapping data structure and for each auction the bids are stored in a list. Once the seller calls the original `closeAuction()` function, the selected winner bid ID is communicated in the parameters and the smart contract has to search in the nested structure, identify the winner and refund all the losers.

The selling strategy introduction enables a better way to handle the data in the smart contract. By knowing the criteria under which a bid results to be better than the others, it is possible to keep track of the current best one only. In the project details, the smart contract receives the strategy information in the `createAuction()` method. When the smart contract receives a new bid, it is able to analyse the bid under the criteria defined by the strategy. If the received bid results to be better than the previous best one, the best bid details are updated and the previous winner is immediately refund. Finally, when the seller call the `closeAuction()` function, the smart contract knows exactly who is the winner and no further computations needs to be performed. The money are transferred to the seller and auction is closed. In this new implementation, the amount of data stored in the blockchain are drastically reduced. The improvements directly correspond to an increased smart contract efficiency and so to a reduction of the costs of the transactions in terms of gas consumption. In particular, it is possible to analyse the computational complexity of the smart contract function and compare them with the original implementation in table 3.1.

The smart contract redesign results in an overall improvement of the execution and more important of the efficiency. From table 3.1 it is possible to notice that in the original implementation the auction life-cycle on the smart contract coincide with $\mathcal{O}(n^2 * m)$ in term of computational complexity (with n the number of auctions and m the number of bids per auction). In DApps the amount of stored data and the computational complexity correspond to a real cost in term of gas consumption. In the original system, an high number of auctions and bids is traduced in a very high transactions costs due to the proportionality with respect to the computational complexity. Instead, the smart contract implementation proposed in the thesis is highly scalable. The number of auctions and bids do not impact on the transactions

costs, in fact the auction life-cycle corresponds in $\mathcal{O}(1)$ in term of computational complexity.

function	original	new
`newUser()`	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`createAuction()`	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`placeBid()`	$\mathcal{O}(n)$	$\mathcal{O}(1)$
`closeAuction()`	$\mathcal{O}(n * m)$	$\mathcal{O}(1)$

Table 3.1: Computational complexity of the smart contract's functions in the original and in the new implementation (n = number of auctions, m = number of bids per auction)

3.3 Front-running

3.3.1 Vulnerabilities

In section 2.4, the front-running attacks on blockchain are introduced and grouped in classes with common characteristics. In this section, the case of the blockchain-based trading system in prosumer communities is analysed in order to highlight the front-running vulnerabilities.

The goal of the thesis is to point out the importance of front-running prevention strategies in the design of the DApp. The analyses has to be performed taking into consideration the attacks described in section 2.4.1 with respect to the presented system. Focusing on the system functioning, it is possible to identify at least one vulnerability for each class of attack:

- in the implemented smart contract if two users bids in the same auction and offers the same amount of money, only the first received one is considered, while the second bid is discarded. If Mallory is a front-runner and sees Alice's pending `placeBid()` transaction, Mallory can perform a *displacement attack* and call the `placeBid()` function in the same state offering the same amount of money before the Alice's pending transaction is mined. As in the definition of the attack provided in section 2.4.1, it is not important that Alice's function call is executed after the front-runner one, because only the attacker bid will be considered. If the seller chose a FIFO strategy and the attack is executed, Mallory (the front-runner) will win the auction and Alice's function call will be orphaned.

- The second considered case is a form of prevention of possible DApp's future changes. In particular, it is possible to imagine an auction-based strategy in which the buyer can only bid for a fixed quantity over the current maximum value of the auction. This kind of strategy is common in many auction-based systems and it can be applied also in the analysed one. If the bid value depends on the current state of the auction the transaction is subject to front-running. For example, if Alice calls a method to place a bid that increase the current value of the auction, Mallory can perform an *insertion attack* and place her bid first. The state of the contract will change and Alice's transaction runs of a modified state of the contract and her offer will depend on Mallory's one.
- In the last situation, the original implementation of the system is considered. The time is divided in time-step and each auction is strictly related to the time-step itself. In fact, at the end of each time-step all the active auctions are closed. If Mallory (the attacker) calls the `placeBid()` function close to the end of the time-step, she can perform a *suppression attack* to delay all the other bids until the time-step ends. The results will be that Mallory wins the auction and all the other bids will be orphaned because the auction will be no more on the market.

The three described attacks highlights the importance of taking into account front-running vulnerabilities during the design phase of the smart contract. Because once the contract is deployed, it cannot be changed anymore and its limits and bugs can only be solved by deploying a new one. Even if the brand new smart contract is deployed on the Ethereum blockchain, the old one still be there and its functions remains callable.

3.3.2 Implemented prevention strategies

Front-running prevention strategies presented in 2.4.3 are divided in three main classes, each of them has to be considered in relation to the project vulnerabilities identified in 3.3.1. The goal of the thesis is to redesign the smart contract implementing the mitigations that best fits with the project needs.

The first category to be analysed is *transaction sequencing*. The idea is to force a sequence to the transactions even if the blockchain does not support it. The considered system is based on Ethereum, where pending transactions are stored in a transaction pool called "mempool" in which there is no intrinsic order. The only way to sequence the transactions, it is by acting on the smart contract. More in details, is possible to define a function that accepts transactions that specify the state of the blockchain as the only state to be executed on. In the presented smart contract, a transaction sequencing front-running mitigation is implemented

in the `placeBid()` function. Once the user place the bid on the user interface, the web application call the smart contract function `getTxReceived()` that returns the number of received bids for the specified auction. The number of received bids corresponds to a state variable for that specific auction. The `placeBid()` function requires that the current state of the auction corresponds to the one received in as parameter. This is a simple approach that can be considered as a prevention strategy for the insertion attack described in 3.3.1 in which the front-runner change the state of the contract before executing the received transaction.

The second category is *confidentiality* which commonly consists in the commit/reveal approach. The two-way communication results in a increased gas to pay for the transactions execution. In the particular case of the analysed system, the gas consumption is a constrain and an increment in this direction can compromise the performance of the entire project. The outcome is that confidentiality is not suitable and cannot be included in the project. It can be considered as a example of the impossibility to standardise front-running mitigations, they must be tailored over each specific smart contract.

Finally, the approach that results to be the best solution for the project is the one that is defined in 2.4.3 as *design practices*. This last category assumes that front-running is unpreventable. The only way to mitigate it is to redesign the smart contract to remove any benefit from it or discourage attackers. The main vulnerability of the presented DApp is the time dependency of the auction-based system, therefore specific prevention strategies need to be tailored in order to do not compromise the DApp functioning.
The first mitigation is the addition of a check in the `placeBid()` function, that now requires that the gasPrice of the received transaction must be under a fixed threshold of 21 Gwei (MetaMask default is 20 Gwei) to be accepted. The focus is not on the the value of the threshold itself, but on its presence. In fact, it is sufficient to select the gasPrice of a transaction equal to the threshold and the transaction would not be subject to any front-running attack based on gasPrice.
The second strategy needs a redesign of the original system functioning. In the original implementation, each auction is closed at the end of the respective time-step. The front-runner can perform a suppression attack right before the end of the time-step in order to win the auction. In the proposed auction management approach, the auction is closed by the seller who can call the `closeAuction()` function at any time regardless of the time-step and the results is that the attacker cannot previously know the amount of time they need to perform a suppression attack.

3.3.3 Cost of the prevention strategies

The thesis purpose is to study front-running vulnerabilities of the considered system and implement tailored attacks mitigations. In order to do that, the system functioning must be maintained and system constraints must be considered.
One of the main restriction imposed by a platform to sell and buy electricity is the cost-effectiveness. If the costs of the transactions overcome the cost of the electricity itself the system usefulness fail.
The challenge is to implement front-running prevention strategies that do not affect the transactions costs, with a minimum impact on the system functioning.

Summing up the implemented prevention strategies, it is possible to underline the minimal impact they have on the system:

- the addition of the `getTxReceived()` function do not affect neither the costs nor the functioning. The method is defined as "view" in the smart contract, which means that only provides data to the web application. No computational operations are performed. Therefore, the function can be called in background and the user do not notice any difference with the previous implementation.
- The included require statement in the `placeBid()` function regarding the gasPrice's maximum value to be accepted is a $\mathcal{O}(1)$ operation which do not affect neither the costs nor the functioning.
- The only change that slightly affect the functioning is the removed time-step dependency of the auction, but the previously described potential beneficial effects are much greater than the adjustment itself.

The implemented strategies to avoid front-running attacks successfully achieve the goal of do not affect the costs and keeping as constant as possible the functioning of the system. In particular, they not increase the computational complexity of the transactions. Thus, the gas consumption remains the same and the overall cost stay constant. This is an important achievement to demonstrate that customised solutions to front-running can be designed to match every kind of faced constraint, even in a complex and innovative system as the analysed one.

Chapter 4

Future improvements

The thesis proposes a new implementation of the blockchain-based trading system for prosumers communities together with new features, increased efficiency and front-running prevention strategies.

An important future improvement, related to the easiness of use, is the development of a mobile application to allow users interacting with the system from their smartphones. This could be a great idea to let people accept the platform, since nowadays it is much more common to do any kind of internet operation from a mobile device than from a computer, especially because smartphones are almost always close to people.
Moreover, an interesting field to explore would be the addition of more selling strategies to best fits with users and market needs. In particular, different strategies could be designed to optimise the overall energy affordability in order to overcome the gap between decentralised renewable production and classical centralised energy provisioning. The proposed implementation of the system is already set up to accept additional strategies, only few changes has to be performed in the code.

Finally, it is reasonable to assume that the platform is ready to be tested in a real world environment in order to measure the effectiveness of the system and establish a communication with real smart meters. Further investigations can be performed in the smart grids direction to better exploit the MatPower functionalities and prosumers hardware interaction to optimise the electricity flow on the physical grid.

Chapter 5

Conclusions

The considered DApp is part of a more complex system that spans across multiple technologies to ensure a communications between all the system actors. It is possible to consider the analysed system as a very particular and innovative case in decentralised applications world, in fact it is linked with physical infrastructures of smart grid via the smart meter and MatPower tool.

The thesis propose a new implementation of the blockchain-based trading system for electricity in prosumers communities with the following main improvements. The fist one is the introduction of a new auction management approach, that enables wider range of market opportunities for the users and it makes the system compatible with any kind of electricity storage equipment. The second improvement consists in the addition of the selling strategies to exploit the blockchain's decentralised transparency, together with the automation of the auction winner selection. Moreover, the smart contract is completely redesigned to increase the efficiency in term of computational complexity which correspond to a real cost for the user. In particular, the auction life-cycle complexity is reduced from $\mathcal{O}(n^2 * m)$ (with n the number of auctions and m the number of bids per auction) to $\mathcal{O}(1)$. The result is a smart contract that does not depend on the number of data stored in the blockchain, therefore a constant transaction cost is ensured.
Finally, the front-running vulnerabilities of the system are identified and the respective mitigations are implemented. Results underline the importance of a case-by-base investigation of front-running in DApps in order to design tailored solutions that do not affect the system functioning and constraints. The main challenge of the presented system is the cost-effectiveness. For this reason, the prevention strategies are studied with the goal of do not affect the transaction costs and thus the computational complexity of the functions calls. Results demonstrate that customised mitigation to front-running can be designed to match any kind of constraint, even in a complex and innovative system as the analysed one.

Appendix A

Smart contract

```
pragma solidity ^0.6.12;

contract ElectricityTrading {
    // map user wrt the WalletAddress
    mapping(address => uint128) private usersByAddress;

    // map user wrt the SmartMeterId
    mapping(uint128 => address) private usersBySMId;

    // map the AuctionId with the respective Auction are
    // indicized with their AuctionId
    mapping(string => Auction) public auctions;

    //21000000000 wei = 21Gwei (MetaMask default is 20Gwei)
    uint256 constant maxGasPrice = 21000000000; // [wei]

    // An Auction contains the strategy and all the bids
    // associated to a seller
    struct Auction {
        address payable sellerWalletAddress;
        uint8 strategy; // 0: FIFO, 1: MaxPrice
        uint256 basePrice; // [wei]
        address payable bestBidAddress;
        uint256 bestBidValue; // monetary value of the bid [wei]
        bool exists;
        uint128 txReceived; // Current state of the auction.
    }

    // constructor
    constructor() public {}

```

```
32
33
34      // Only a registered user can interact.
35      modifier onlyUsers {
36          require(usersByAddress[msg.sender] != 0,
37              "User not allowed");
38          _; // otherwise, the function is executed
39      }
40
41      // Register a new user:
42      // - WalletAddress -> SmartMeterId
43      // - SmartMeterId -> WalletAddress
44      function newUser(uint128 _smartMeterId) external {
45          // the new user must not be registered yet
46          require(usersByAddress[msg.sender] == 0,
47              "Wallet already registered");
48          require(
49              usersBySMId[_smartMeterId] == address(0),
50              "Smart Meter Id already registered"
51          );
52
53          // register user
54          usersByAddress[msg.sender] = _smartMeterId;
55          usersBySMId[_smartMeterId] = msg.sender;
56      }
57
58      // Create an instance of the auction structure auctions
59      function createAuction(
60          string memory _auctionId,
61          uint8 _strategy,
62          uint256 _basePrice
63      ) external onlyUsers {
64          // auctionId does not exist yet
65          require(auctions[_auctionId].exists != true,
66              "Auction already created");
67          // strategy is implemented
68          require(_strategy == 0 || _strategy == 1,
69              "Strategy not available");
70
71          // create a new auction
72          auctions[_auctionId] = Auction({
73              sellerWalletAddress: msg.sender,
74              strategy: _strategy,
75              basePrice: _basePrice,
76              bestBidAddress: address(0),
77              bestBidValue: 0,
78              txReceived: 0,
79              exists: true
80          });
```

```
81      }
82
83      function getTxReceived(string memory _auctionId)
84          external
85          view
86          returns (uint256 txReceived)
87      {
88          // define a pointer for the auction
89          Auction storage auction = auctions[_auctionId];
90
91          // auctionId must exists
92          require(auction.exists == true,
93              "Auction do not exists");
94          return auction.txReceived;
95      }
96
97      // Place a bid.
98      // - check the requirements
99      // - receive the respective value
100     // - save as the best received offer
101     function placedBid(string memory _auctionId, uint128 _txReceived)
102         external
103         payable
104         onlyUsers
105     {
106         // define a pointer for the auction
107         Auction storage auction = auctions[_auctionId];
108
109         // [Front-Running] design choice to fix a max gas price
110         // to limit the possible front-runner
111         require(
112             tx.gasprice < maxGasPrice * 1 wei,
113             "Gas price exceeded
114             (design choice to avoid front-running)"
115         );
116
117         // auctionId must exists
118         require(auction.exists == true,
119             "Auction do not exists");
120
121         // [Fron-running]
122         require(
123             auction.txReceived == _txReceived,
124             "The state of the auction has changed"
125         );
126
127         // check that the msg.value is at least equal
128         // to the basePrice of the auction
129         require(
```

```
130             msg.value >= auction.basePrice * 1 wei,
131             "The value of the bid is lower than
132             the base price of the auction"
133         );
134
135         require(
136             msg.value > auction.bestBidValue * 1 wei,
137             "The value of the bid is lower or equal than the
138             current best bid"
139         );
140
141         auction.txReceived++;
142
143         if (auction.strategy == 0) {
144             // if FIFO strategy
145             auction.bestBidAddress = msg.sender;
146             auction.bestBidValue = msg.value;
147
148             // close the auction since the strategy is FIFO
149             closeAuctionInternal(_auctionId);
150         } else if (auction.strategy == 1) {
151             // MaxPrice strategy
152             if (msg.value > auction.bestBidValue) {
153                 // if the 2 values are equal
154                 // the winner is the first to arrive
155                 // new best offer received
156                 // refund the old best bidder
157                 auction.bestBidAddress.transfer(
158                     auction.bestBidValue);
159
160                 // Update the best offer
161                 auction.bestBidAddress = msg.sender;
162                 auction.bestBidValue = msg.value;
163             }
164         }
165     }
166
167     // Close the auction (internal call from "placeBid" function):
168     // - send money to the seller
169     // - remove the auction from the smart contract
170     function closeAuctionInternal(string memory _auctionId)
171     internal {
172         // define a pointer for the auction
173         Auction storage auction = auctions[_auctionId];
174
175         // addresses must be equal and assigned
176         assert(
177             msg.sender == auction.bestBidAddress &&
178                 auction.bestBidAddress != address(0)
```

```
179         );
180
181         // send money to the seller
182         auction.sellerWalletAddress.transfer(
183             auction.bestBidValue);
184
185         // delete the auction
186         delete auctions[_auctionId];
187     }
188
189     // Close the auction:
190     // - send money to the seller
191     // - remove the auction from the smart contract
192     function closeAuction(string memory _auctionId) public payable
        onlyUsers {
193
194         // define a pointer for the auction
195         Auction storage auction = auctions[_auctionId];
196
197         // auctionId must exists
198         require(auction.exists == true,
199             "Auction do not exists");
200
201         // the auction must be owned by the transaciton's sender
202         require(
203             msg.sender == auction.sellerWalletAddress,
204             "You can only close your auctions"
205         );
206
207         // at least one bid must be received
208         require(auction.bestBidAddress != address(0),
209             "No received bids");
210
211         // send money to the seller
212         auction.sellerWalletAddress.transfer(auction.bestBidValue);
213
214         // delete the auction
215         delete auctions[_auctionId];
216     }
217 }
218 // end of the contract
```

Bibliography

[1] Eunice Espe, Vidyasagar Potdar, and Elizabeth Chang. «Prosumer Communities and Relationships in Smart Grids: A Literature Review, Evolution and Future Directions». eng. In: *Energies* 11.10 (2018), p. 2528. ISSN: 1996-1073. URL: https://doaj.org/article/9aa419c97f014b248e29051812714dfc (cit. on p. 1).

[2] Axel Gautier, Julien Jacqmin, and Jean-Christophe Poudou. «The prosumers and the grid». eng. In: *Journal of Regulatory Economics* 53.1 (2018), pp. 100–126. ISSN: 0922-680X (cit. on p. 1).

[3] «Directive (EU) 2018/2001 of the European Parliament and of the Council of 11 December 2018 on the promotion of the use of energy from renewable sources». In: *Official Journal of the European Union* (2018) (cit. on p. 2).

[4] Campos Inês, Pontes Luz Guilherme, Marín-González Esther, Gährs Swantje, Hall Stephen, and Holstenkamp Lars. «Regulatory challenges and opportunities for collective renewable energy prosumers in the EU». eng. In: *Energy Policy* 138 (2020). ISSN: 0301-4215 (cit. on p. 2).

[5] Jerry W Markham. «Front-running-insider trading under the commodity exchange act». In: *Cath. UL Rev.* 38 (1988), p. 69 (cit. on p. 3).

[6] *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf. [Online; accessed 14-May-2020] (cit. on p. 5).

[7] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang. «Untangling Blockchain: A Data Processing View of Blockchain Systems». In: *IEEE Transactions on Knowledge and Data Engineering* 30.7 (2018), pp. 1366–1385 (cit. on p. 5).

[8] Mark Comerro. «Blockchain e Smart Contracts finalizzati a gare d'appalto». MA thesis. Italy: Politecnico di Torino, 2018 (cit. on pp. 6, 8, 9, 11).

[9] Andreas M. Antonopoulos. *Mastering Bitcoin*. 1005 Gravenstein Highway North, Sebastopol, CA 95472, United States of America: O'Reilly Media, 2017 (cit. on pp. 7, 9).

[10] Chris Dannen. *Introducing Ethereum and solidity*. Vol. 1. Springer, 2017 (cit. on p. 10).

[11] Valentina Gatteschi, Fabrizio Lamberti, Claudio Demartini, Chiara Pranteda, and Victor Santamaria. «To Blockchain or Not to Blockchain: That Is the Question». eng. In: *IT Professional* 20.2 (2018), pp. 62–74. ISSN: 1520-9202 (cit. on pp. 11, 12, 22).

[12] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. «Sok: Transparent dishonesty: front-running attacks on blockchain». In: *International Conference on Financial Cryptography and Data Security*. Springer. 2019, pp. 170–189 (cit. on pp. 13–16).

[13] *MetaMask*. `https://chrome.google.com/webstore/detail/metamask/nkbihfbeogaeaoehlefnkodbefgpgknn?hl=en`. [Online; accessed 11-May-2020] (cit. on p. 21).

[14] *Ganache*. `https://www.trufflesuite.com/ganache`. [Online; accessed 11-May-2020] (cit. on p. 21).

[15] *Calling Matlab from Python*. `https://it.mathworks.com/help/matlab/matlab-engine-for-python.html`. [Online; accessed 13-May-2020] (cit. on p. 21).

[16] R. D. Zimmerman and C. E. Murillo-Sanchez. *MATPOWER User's Manual, Version 7.0*. 2019. URL: `https://matpower.org/docs/MATPOWER-manual-7.0.pdf` (cit. on pp. 21, 23).

[17] *CherryPy — A Minimalist Python Web Framework*. `https://docs.cherrypy.org/en/latest/`. [Online; accessed 13-May-2020] (cit. on p. 21).

[18] *What Is MongoDB?* `https://www.mongodb.com/what-is-mongodb`. [Online; accessed 14-May-2020] (cit. on p. 21).

[19] *Express4.17.1: Fast, unopinionated, minimalist web framework for Node.js*. `https://expressjs.com/`. [Online; accessed 14-May-2020] (cit. on p. 22).

[20] *React: A JavaScript library for building user interfaces*. `https://reactjs.org/`. [Online; accessed 14-May-2020] (cit. on p. 22).

[21] *About Node.js*. `https://nodejs.org/en/about/`. [Online; accessed 14-May-2020] (cit. on p. 22).

[22] Chris Dannen. *Introducing Ethereum and Solidity*. Jan. 2017. DOI: `10.1007/978-1-4842-2535-6` (cit. on p. 22).

[23] Digiconomist. *Ethereum Energy Consumption Index (beta)*. `https://digiconomist.net/ethereum-energy-consumption`. [Online; accessed 11-May-2020] (cit. on p. 22).

[24] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas. «MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education». In: *IEEE Transactions on Power Systems* 26.1 (2011), pp. 12–19 (cit. on p. 23).