AN ABSTRACT OF A DISSERTATION

BLOCKCHAIN-BASED SECURE AND PRIVACY-PRESERVING SCHEMES FOR CONNECTED VEHICLES

Mohamed Baza

Doctor of Philosophy in Electrical and Computer Engineering

Nowadays, blockchain and smart contract technology is gaining massive attention due to the security features it brings. In this dissertation, we leverage this technology to design secure and privacy-preserving schemes for connected vehicles applications.

We first propose a decentralized firmware update dissemination scheme for the autonomous vehicles (AVs). The scheme enables AVs, namely distributors, to participate in the distribution process of the updates to guarantee high availability and fast delivery. To incentivize AVs to distribute the updates, a reward system is developed to maintain a credit reputation for each distributor AV. A zero-knowledge proof protocol is used to enable AVs to exchange a firmware update in return for a proof of distribution that is used to update the AV reputation.

Then, we propose a decentralized privacy-preserving ride-sharing organization scheme using public blockchain. To discourage submitting multiple ride requests or offers, while not committing to any of them, we develop a time-locked deposit protocol where drivers and riders lock a deposit to a smart contract. Later, a driver has to prove to the blockchain that he/she arrived the pick-up location on time to get the deposit, otherwise the rider obtains it. In addition, we introduce a reputation model to rate drivers based on their past behavior.

Then, we design a decentralized charging coordination mechanism for Energy storage Units (ESUs). The idea is that ESUs send anonymous charging requests to a smart contract. Then, the contract runs a charging coordination mechanism such that ESUs with the highest priority indices are charged in the present time slot while charging requests of lower-priority ESUs are deferred to future time slots.

Finally, we propose a blockchain-based privacy-preserving schemes for the energy trading of Electric Vehicles (EVs). Launching Sybil attacks, e.g., to submit a large number of messages and pretending that they are submitted from different EVs to launch Denial of service (DoS) attacks, are thwarted by our schemes to ensure that the energy trading system is reliable and the service is available. To preserve the privacy of EV owners, an efficient anonymous payment system is developed to allow EVs to pay their charging fees with untraceable digital coins.

BLOCKCHAIN-BASED SECURE AND PRIVACY-PRESERVING SCHEMES FOR CONNECTED VEHICLES

A Dissertation

Presented to

the Faculty of the College of Graduate Studies

Tennessee Technological University

by

Mohamed Baza

In Partial Fulfillment

of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

Electrical and Computer Engineering

December 2020

CERTIFICATE OF APPROVAL OF DISSERTATION

BLOCKCHAIN-BASED SECURE AND PRIVACY-PRESERVING SCHEMES FOR CONNECTED VEHICLES

by

Mohamed Baza

Graduate Advisory Committee:

Mohamed Mahmoud, Chairperson	Date
Doug Talbert	Date
Ambareen Siraj	Date
Syed Rafay Hasan	Date
Ghadir Radman	Date

Approved for the Faculty:

Mark Stephens
Dean
College of Graduate Studies

Date

DEDICATION

This thesis is dedicated to my mother, my wife and my great family. Without their endless love and encouragement I would never have been able to complete my graduate studies.

ACKNOWLEDGMENTS

My profound gratitude goes to the Almighty God for the knowledge and wisdom he has bestowed upon me to complete this thesis. I'm grateful to all the people who were helpful in the creation of this thesis. Thanks to my committee members, Dr. Mohamed Mahmoud, Dr. Doug Talbert, Dr. Ambareen Siraj, Dr. Syed Rafay Hasan and Dr. Ghadir Radman for their help and support. Thanks to my colleagues, friends, and well wishers for all their valuable contribution. Finally, I thank my family for their prayers, patience, support, and encouragement all through the duration of my graduate study.

TABLE OF CONTENTS

Page

Chapter Page

Chapter Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this section, we first give an overview on blockchain and smart contract technology, followed by the problem formulations of our dissertation.

1.1 Blockchain Technology

Blockchain is a transparent data structure that is organized as a chain of blocks and managed by a network of computers, called miners, running a peer-to-peer (P2P) protocol. Each block contains a set of transactions that are committed by network peers according to a predefined consensus algorithm [1]. Blockchain was first introduced as a distributed cryptocurrency that enables the transfer of electronic cash without the intervention of banks. Since then it has evolved beyond that to support the deployment of more general-purpose distributed applications. This concept has been introduced by Vitalik Buterin and is called smart-contracts or decentralized autonomous organizations [2]. A smart-contract can be described as an autonomous computer program running on blockchain network. This program acts as a contract whose terms can be pre-programmed with the ability of self-executing and self-enforcing itself without the need for trusted authorities [3, 4].

A typical blockchain system consists of multiple nodes which do not fully trust each other. Together, the nodes maintain a set of shared global states, and perform transactions that may modify the states. Blockchain is a special data structure which stores historical states and transactions. All nodes in the system agree on the transactions and their order. Fig. 1.1 shows the blockchain data structure, in which each block is linked to its predecessor via a cryptographic pointer, all the way back to the first (genesis) block. Because of this, blockchain is often referred to as a distributed ledger. The block contain the roothash (or Merkle root) which is the hash of all the hashes of all

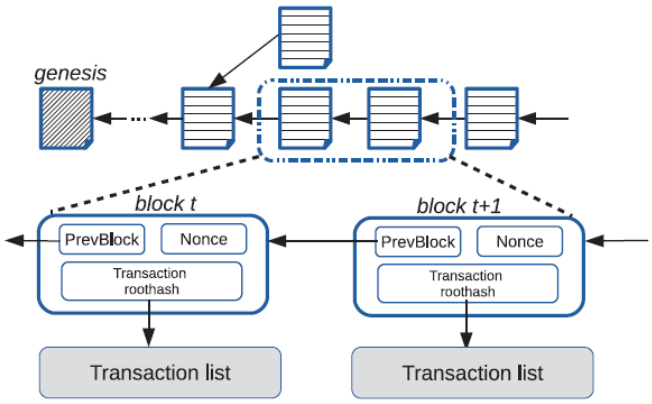


Figure 1.1: Illustration of a blockchain structure, where transactions are packed into blocks that are linked to previous blocks.

the transactions [3]. For the block to be added to the chain, a miner has to successfully solve a computationally hard puzzle (finding the right nonce for the block header).

1.1.1 Features of Blockchain

Blockchain offers several key features such as:

1. *Decentralized data management.* Every peer in the system has the authority to add data to the ledger, in other words make transactions, so no one user owns the system more than any other.

2. *Data security, tamper-proof, anti-forgery and data integrity.* Blockchain is architectured to store data such that it is immutable and tamper-proof. The decentralized nature of blockchain makes it overly challenging for attackers to modify the ledger.

3. *Distributed ledger and transparency.* A shared public list of transactions (the exchange of data) allows every peer in the network to have access to every transaction made, making the system transparent.

4. *No central piont of failure.* The lack of a centralized storage system eliminates the risk of losing the stored data by compromising one node.

5. *High efficiency.* Checking balances and completing transactions in a blockchain system can, in theory, be instantaneous.

1.1.2 Public versus Private Blockchains

At a high level, a blockchain system can be categorized as either public or private. In the former, any node can join and leave the system, thus the blockchain is fully decentralized, resembling a peer-to-peer system. In the latter, the blockchain enforces strict membership. More specifically, there is an access control mechanism to determine who can join the system, so, every node is authenticated and its identity is known to the other nodes.

Public Blockchains. Bitcoin is the most well known example of public blockchains. In Bitcoin the states are digital coins (cryptocurrencies), and a transaction moves coins from one set of addresses (for accounts) to another. The idea is that each node broadcasts a set of transactions it wants to perform. Special nodes, called miners, collect transactions into blocks, check for their validity, and start a consensus protocol to append the blocks onto the blockchain. Bitcoin uses proof-of-work (PoW) for consensus, where only the miner which can successfully solve a computationally hard puzzle (finding the right nonce for the block header) can append the block to the blockchain. Also, Etherum is another well known example for the public blockchains and it is the second-largest cryptocurrency platform after bitcoin. It has its own cryptocurrency that is Ether.

Private Blockchain. Hyperledger [5] is among the popular private blockchains. Since node identities are known in the private settings, most blockchains adopt one of the protocols from the vast literature on distributed consensus. Zab [6], Raft [7], and PBFT [8] are popular consensus protocols that are widely used nowadays. Hyperledger directly uses PBFT consensus that has three-phases.

```
contract Doubler {
  struct Partitipant {
    address etherAddress;
    uint amount;
  }
  Partitipant[] public participants;
  uint public balance = 0;
  ...
  function enter() {
    ...
    balance+= msg.value;
    ...
    if (balance >
              2*participants[payoutIdx].amount) {
      transactionAmount = ...
      participants[payoutIdx].
              etherAddress.send(transactionAmount);
      ...
    }
  }
  ...
}
```

Figure 1.2: An example of Ethereum smart contract written in Solidity, which implements a pyramid scheme.

In the *pre-prepare* phase, a leader broadcast a value to be committed by other nodes. Next, in the *prepare* phase, the nodes broadcast the values they are about to commit. Finally, the *commit* phase confirms the committed value when more than two third of the nodes agree in the previous phase.

1.1.3 Smart Contracts

A smart contract refers to the program executed when a transaction is performed. It can be regarded as a stored procedure invoked upon a transaction. The inputs, outputs and states affected by the smart contract execution are agreed on by every node. All blockchains have built-in smart contracts that implement their transaction logics. In crypto-currencies, for example, the built-in smart contract first verifies transaction inputs by checking their signatures. Next, it verifies that the balance of the output addresses matches that of the inputs. Finally, it applies changes to the states. One way to classify smart contracts is by its language.

Ethereum smart contracts can specify arbitrary computations, i.e., they are Turing complete code. Fig. 1.2 shows a snippet of a real smart contract running on Ethereum. It implements a

pyramid scheme in which users send money to the contract that pays interests to early participants. The contract has its own states, namely the list of participants, and exports a function called *enter*. A user invokes the contract by sending money through a transaction. When executed, the contract can access the input address (user account) via *msg.sender* and the transaction value via *msg.amount*. Then, it updates the accumulated balance and computes the interest for each participants. Finally, payment is made by invoking *etherAddress.send*.

1.2 Problem Formulation

In this dissertation, we leverage blockchain and smart contract technology to design secure and privacy-preserving schemes for connected vehicles applications. Specifically, we consider the following applications: firmware update dissemination of autonomous vehicles (AVs), ride sharing organization, charging coordination of energy storage units (ESUs), and energy trading system.

1.2.1 Firmware Update Dissemination for Autonomous and Connected Vehicles

Over the last few years, the automobile industry has achieved a notable leap towards the realization of practical AVs [9, 10, 11]. AVs are equipped with sophisticated systems and subsystems to provide vehicles with advanced communication capabilities, computer vision, autonomous decision-making capability, etc., to enable them to autonomously drive without any human intervention [12]. AVs have the potential to revolutionize our current transportation system by reducing congestion and travel time, increasing fuel efficiency, and improving road safety [13, 14, 15, 16, 17].

AVs are composed of many subsystems running specific firmware programs that enable performing all control, monitoring, and data manipulation operations. However, by controlling the functionality of the subsystems through the installation of infected versions of the corresponding firmwares, an attacker can successfully hack AVs and fully/partially control them, e.g., to involve the vehicle in accidents deliberately, which may lead to financial loses and losing lives. As an ex-

ample of this attack, Chrysler company announced a recall of 1.4 million vehicles after hackers had managed to turn-off the engine remotely while the vehicles were on motion by exploiting a hackable software vulnerability via the internet-connected entertainment system [18]. Therefore, ensuring the integrity and authenticity of AVs' firmware update is primordial and must be carefully investigated. In addition, it may happen that multiple AVs with their various subsystems need to be updated urgently and fast, e.g., to fix newly discovered bugs, thus the high availability of the updates is required.

Most of the existing solutions for firmware update depend on the client-server model in which a manufacturer delegates the process of firmware distribution to trusted cloud providers, such as Microsoft Azure and IBM Cloud [19]. However, this central client-server architecture suffer from the single point of failure and attack. In case the server is not available, the clients (AVs) cannot access the resources (updates). For AVs, there are several factors that make the availability and security of the firmware updates a challenging task. To elaborate, the number of autonomous vehicles on roads is expected to reach 20.8 million in the U.S. alone [20]. Also, each AV has many subsystems that run different programs designed to accomplish specific functions. This creates tremendous load on the server side and can broaden the sources of cyberattacks. Moreover, vehicles may last for many years (more than 20 years); thus the integrity and authenticity of the AVs' firmwares should be guaranteed for very long time.

Instead of depending on the client-server model, we aim to decentralize the firmware update process by allowing the AVs to participate in the distribution of each new update. Indeed, decentralizing the firmware update process has several advantages such as: (i) high availability of the firmware updates compared to the current client-server architecture and (ii) quick distribution of the updates. However, decentralizing the firmware update process can bring new challenges that should be well addressed in order to ensure a secure firmware update process. These challenges can be listed as follows.

- How to incentivize AVs to participate in the distribution process?
- How to ensure the integrity of the update in trust-less environment?

To illustrate, assume we have two vehicles A and B that do not trust each other. A has a new release of an update and wants to share this update with B. However, A needs to make sure that it will be compensated by some incentive, meanwhile, B needs to ensure that the update it receives is authentic. To sum up, a secure and decentralized firmware update scheme should achieve the following:

- Availability. The scheme should ensure high availability compared to the traditional client-server model.
- Incentivization. The scheme should incetivize participants to distribute the updates.
- Integrity. The scheme should enable receivers of new updates to ensure the integrity of the updates.

1.2.2 Ride Sharing with Privacy-Preservation, Trust, and Fair payment atop public Blockchain

Over the last few years, technolgy has changed our life [21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35], for example ride-sharing services (RSSs) have been emerging as an alternative transportation service that allows people to use personal cars more efficiently. In RSSs, a driver shares his vacant car seats with other riders [11, 36, 9, 37]. Ride sharing has several benefits to the individuals and the community at large including increasing occupancy rates, sharing travel costs, extending social circles, and reducing both fuel consumption and air pollution [38, 39]. Across the world, many providers that offer online ride-sharing services such as Flinc, UberPool, Lyft Line and Blablacar have emerged. According to [40], the ride sharing market is projected to reach USD 218 billion by 2025.

A ride-sharing service can find the drivers and riders who can share rides by matching drivers' ride offers (i.e., planned trips) to riders' requests (i.e., desired trips). To enable ride-sharing service, users (i.e., drivers and riders) have to share with a service provider the trip information, including departure time, location, and destination. The service provider works as a middleman to facilitate the communication between the system users and usually charges a commission for each successful shared ride. However, running the service by a central server, makes the system vulnerable to a single point of failure and attack [41]. Moreover, the system is subject to data privacy breach by the service providers themselves or external attackers. BlaBla car privacy policy (in https://blog.blablacar.co.uk/about-us/terms-and-conditions) states [42]:

"If you are a rider, we may collect location information when the app is running in the foreground. In certain regions, may also collect this information when the app is running in the background of your device if this collection is enabled through your app settings or device permissions."

In addition, if the security of the service provider is compromised, the service can be interrupted and the data can be disclosed, altered, or even deleted. For instance, Uber has witnessed a tremendous data leakage of 57 million customers and drivers for more than a year. Uber has paid 148 million just to settle an investigation to its data breach [43]. Similarly, in April 2015, due to hardware failure in Uber China, a service outage occurred and passengers were not able to complete their orders at the end of services [44]. In addition, in order to maximize their benefits, most ride-sharing service providers impose a high service fee that can reach up to 20% [45].

In contrast to the traditional client-server model, our goal is to propose a blockchain-based ride-sharing scheme to mitigate the single point of failure issues presented in the client-server architectures. However, besides being completely distributed and transparent, the openness of blockchain leads to a potential privacy concern where the data can be publicly accessible. Despite the use of anonymous authentication, this is not sufficient to protect the privacy of the end users. For instance,

by tracking the activity of a driver or rider, an attacker with little background knowledge of that user can figure out all his location traces [46]. Moreover, because in public blockchains, anyone can join and transact in the network anonymously, malicious users can disturb the blockchain-based ride-sharing service by sending, for instance, multiple requests/offers while not committing to them. Therefore, it is required to keep track of drivers' behaviours and build a reputation system that helps a rider to select with confidence an appropriate driver for his/her ride request. Subsequently, in order to decentralize the ride-sharing services efficiently, the privacy concerns need to be carefully studied and addressed. This mainly requires achieving two conflicting objectives, i.e., (i) the desire to have a *transparent* system while protecting users *privacy*, and (ii) ensure *accountability* while being *anonymous*.

1.2.3 Charging Coordination Schemes for Energy Storage Units

Energy storage units (ESUs), including home batteries and electric vehicles (EVs), will play a major role in the future smart grid [47, 48]. They can store energy when there is a surplus in energy generation and inject energy to the grid when the demand is high to balance the energy demand and supply, which in turn enhances the power grid resilience [49, 50]. The ESUs can also facilitate the use of renewable energy generators by storing the excess energy generated. Moreover, ESUs can help electricity consumers to reduce their electricity bills by charging from the grid during low-tariff periods and power houses during high-tariff periods.

Despite their benefits, ESUs pose several challenges that need to be addressed for efficient integration in the power grid. In specific, a simultaneous mass-scale uncoordinated charging of ESUs may lead to lack of balance between the charging demand and the energy supply resulting in instability of the grid [51]. In severe cases, this could lead to a blackout. In order to mitigate such consequences, there is a substantial need for a charging coordination mechanism to avoid stressing the distribution system and prevent power outage [52]. In a charging coordination mechanism,

ESUs should report data such as the time-to-complete-charging (TCC), the battery state-of-charge (SoC), and the amount of required charging. Then, a trusted party, i.e, a charging controller, selects charging requests with high priorities to charge and defer others to another time slot without exceeding the available charging power.

However, the existing charging coordination mechanisms [53, 48] suffer from several limitations. First, they rely on a single entity, namely charging coordinator (CC), to coordinate the charging requests. In turn, this can lead to the single server point of failure and attack, i.e., if a successful denial of service attack is launched on the CC, it will be down and consequently a large number of charging requests could not be coordinated. Second, most of the existing works consider that the CC is a trusted party which is completely honest in scheduling charging requests. As a result, the ESU owners are not aware whether the charging schedules are computed correctly. In specific, the existing charging coordination mechanisms are not transparent. Third, the existing charging coordination mechanisms require that the ESUs report some data to the CC such as whether an ESU needs to charge or not, the TCC, the battery SoC, and the amount of required charging. This, in turn, reveals private information about the owners of the ESUs, such as the location of an EV and the activities of a house's residents [54, 55]. For example, the charging demands sent from an EV can reveal whether the EV's owner is at home and how long he/she will stay, and how often he/she drives. Also, if a home battery is not charged for an extended period, this can reveal that the residents do not spend time at home because they are traveling.

In this research, based on the aforementioned limitations, our goal is to design a charging coordination scheme for ESUs that is decentralized, transparent, and privacy-preserving.

1.2.4 Privacy-Preserving Blockchain-based Energy Trading Schemes for Electric Vehicles

New advancement in technology has made our life much easier [56, 57, 58, 59, 60, 61]. One of these advancement is the smart grid, which is a revolutionary upgrade to the traditional electricity grid that aims to create a clean, resilient, and efficient system by integrating information technology, data communication, sensing, and control technologies into the power system [62, 50, 63]. The smart grid promotes a high penetration level of renewable energy sources and EVs to reduce greenhouse gas emissions [64, 65, 66]. Specifically, residents can install solar panels on their rooftops to generate electricity to power their homes and charge their EVs. EVs are becoming increasingly popular and it is expected that most of the vehicles will be electric in the future [67].

EVs can be charged at home from electricity grid and solar panels [68]. Another charging approach is to charge from charging stations (CSs). This form of charging is called charging stations to vehicle (CS2V) energy trading, where energy traders can offer competitive prices to the EVs [69]. CS2V is useful for providing fast charging or when an EV is travelling for long distance. Alternatively, EVs with surplus energy can charge other EVs, and this form of charging is called vehicle-to-vehicle (V2V) energy trading [70]. This approach is useful when charging stations are not available, e.g., in developing countries or remote areas, they are far or do not have sufficient energy resources, or when cheap prices can be offered by the EVs because they can charge from cheap renewable energy sources.

To facilitate charging of EVs, an energy trading system is needed to match the bids of energy sellers to the requests of buyers and connect them. In these systems, a central system (server), i.e., service manager is usually used. To organize the energy trading, both the energy sellers and buyers need to exchange sensitive information with the service manager about the location and time of energy trading. Without any privacy protection, the untrustworthy server and malicious adversaries

can infer sensitive information about the EVs' owners. For example, if the charging stations are installed at the medical clinic parks, working places or hospitals, sensitive information about the EVs' owners can be revealed such as their habits, workplaces, and health conditions. Moreover, the server can infer if the users are on travel or not by monitoring their trading activities, e.g., if they do not buy or sell energy for a long time. Furthermore, it has been shown that the server can infer the real identity of EV drivers using the locations visited by them [71]. Moreover, running the service by a central server, makes the system vulnerable to the single point of failure and cyber attacks [41]. Also, the centralized platform suffers from a lack of transparency since the server can favor certain buyers to be served first and certain sellers to sell their energy first. Furthermore, if the security of the service manager is compromised, the service can be interrupted and the users' data can be disclosed, altered, or even deleted. For instance, Uber has witnessed a tremendous data leakage of 57 million customers and drivers. In addition, relying on existing payment systems (such as credit or debit cards) for trading transactions may violate the privacy of the EV owners, because their charging locations can be known or tracked over long periods of time. On the other hand, utilizing cryptocurrencies such as Bitcoin and Zcash [72] is not a viable solution, since they are prohibited or restricted in some countries [73].

Instead of using a central architecture for the energy trading system, there are some works that used blockchain to design energy trading systems [74, 70, 75, 76] but they mainly focus on the optimization techniques to maximize the sellers' profit. Very few works, such as others [74, 69], have studied the security and privacy issues in the energy trading. In [74], the authors use smart contracts to dynamically select the best bids from various charging stations. The EVs send their planned routes and battery status to the blockchain and then charging stations offer their prices so that the EVs select best offered price. Nevertheless, the proposed scheme does not consider privacy and charging reservations. In [69], EVs post charging requests on the blockchain and then charging stations respond with offers. Then, an EV selects a bid and then posts a commitment that has the

hash of selected charging station identifier and a random number to the blockchain and then when it reaches the charging station, it has to prove that it is the one that did the reservation by revealing the random number included in the commitment. However, the scheme suffers from several problems and limitations. First of all, the selected charging stations do not know if they are selected since the reservation is made hidden on the blockchain and this may lead to scheduling conflict since two different EVs may select the same bid. In addition, the paper does not consider the V2V energy trading case where the privacy of both charging/discharging EVs should be protected. Finally, the paper does not integrate an anonymous payment method into the scheme to enable payment while protecting the privacy of the EVs' owners.

In this research, our goal is to design a secure and privacy-preserving energy trading system that can tackle the aforementioned challenges.

1.3 Dissertation Organization

The remainder of this dissertation is organized as follows. Related works are discussed in chapter 2. In chapter 3, we discuss an incentivized and secure blockchain-based firmware update and dissemination for AVs. In chapter 4, we discuss a ride sharing organization scheme with privacy-preservation, trust and fair payment atop public blockchain. In chapter 5, we discuss a blockchain-based charging coordination mechanism for smart grid energy storage units. In chapter 6, we discuss a privacy-preserving blockchain-based energy trading schemes for electric vehicles. Finally, conclusions and future works are discussed in chapter 7.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we review the existing schemes in securing the firmware update dissemination, ride sharing organization, charging coordination of ESUs, and energy trading of EVs.

2.1 Firmware Update Schemes

Technology has been changed our life [77, 78, 79, 80, 81, 82, 83, 84, 85, 84, 86, 87, 88, 89, 90]. With the increase of the intelligence level of modern vehicles, more computer programs are used int the vehicle. In the literature, the security of autonomous vehicles has been studied in different works [91, 92, 93, 94]. More specifically, the firmware update dissemination has been discussed in several contexts, including wireless sensor network [95, 96], IoT [97, 98], vehicular network [99], etc. The existing works can be classified into centralized (client-server model) or decentralized. In the following we review some of the existing solutions in both classes.

In [99], Nilsson et al. have proposed a firmware update protocol for modern intelligent vehicles over the internet. The authors suggested a client-server method using a web portal that delivers the firmware update in fragments. The fragments are protected using hash chain to ensure the integrity of updates. However, the system is vulnerable to DoS attacks as it relies on a central server.

In sensor networks, several schemes, such as [95], [96], have been proposed to improve the reliability of delivering new updates/security patches by ensuring their integrity. However, these schemes depend on a single entity to distribute the firmware updates, which is not scalable in case of large networks.

In [97], the authors have proposed a decentralized scheme based on a permission-less blockchain to ensure the integrity of updates by having multiple verification nodes instead of depending on a pri-

vate centralized vendor network. For the distribution of updates, a peer-to-peer file sharing network such as BitTorrent is proposed to ensure integrity and versions tractability of updates. Boudguiga et al. [100] have enhanced the previous work by adding a trusted checking node to verify the update before deploying it to IoT devices. However, the scheme does not provide any incentive for devices to participate and distribute firmware updates to others.

In [98], the authors have proposed a software update framework for IoT devices. The framework allows distributors to deliver the updates in return for digital currency paid by the vendor. However, the scheme incurs high financial cost since it depends on Etherum blockchain [101] which applies fees for each transaction sent to the network.

Although there are some works that have been proposed to use the blockchain technology instead of the client-server model to ensure integrity of the firmware updates, there is no incentive mechanism has been proposed to ensure the availability of the firmware updates. In this dissertation, we leverage the mobility of AVs so that the distributors can exchange the updates with others in return for a proof of distribution and then they get rewarded with an increase of their reputation scores on the blockchain.

2.2 Ride Sharing Organization Schemes

There are several key technologies [102, 103, 104, 105, 106] and ride sharing is one of them that has gained massive interest. In a ride sharing, travelers share a vehicle with others who have the same routes and time schedules. Ride sharing has several advantages such as saving travel cost due to sharing the trip cost, saving fuel, alleviating air pollution and alleviating the traffic congestion [107]. Therefore, ride-sharing has received a lot of attention in the literature [108]. In that context, security and privacy in the centralized setting (client-server model) of ride-sharing service is discussed in [109, 110]. Some companies have developed a blockchain based ride sharing platform e.g., DACSEE [111] Arcade City [112], but without considering location privacy or anonymity. In

the following, we review some of the existing schemes.

In the centralized-setting, in [113], SRide has been proposed to match drivers's offers and riders's requests while protecting the privacy of users against both the service provider and curious users. The service provider uses a filtering protocol based on homomorphic arithmetic secret sharing and secure two-party equality test to determine the subset of drivers with whom the rider can share ride [1].

In [121], Ni et al. have proposed an anonymous mutual authentication protocol by utilizing the BBS+ signature [122] so that drivers and potential riders can mutually authenticate each other without disclosing their real identities. The proposed scheme can also identify and trace a passenger's/driver's identity by a trusted party if a rider/driver complains about the misbehavior of the rider/passenger. Also, the proposed scheme uses plaintext information that includes the pick-up time, locations and price to match riders with drivers, which is a privacy breach for the system users.

Sherif et al. [123] have proposed a centralized privacy-preserving ride sharing scheme. The proposed scheme includes three sub-schemes in which a group signature scheme is used to ensure user anonymity and a similarity measurement technique is used to preserve privacy of users. The idea is that the ride sharing area is divided into cells and each cell is represented by one bit in a binary vector. Then, each user transforms his trip data into a binary vector and submits an encryption of that vector to the cloud server. The cloud server can then measure the similarity of the users' encrypted trip data to determine if a driver can share a ride with a rider without knowing the plaintext data.

In [39], a ride sharing scheme has been proposed. The idea is to use a peer-to-peer ride-sharing management network like u-torrent instead of the service provider where the peers are the

[1] [114, 115, 116, 117, 118, 119, 120]

drivers and riders themselves. Drivers and riders rate each other at the end of each shared trip. However, the scheme does not consider privacy of drivers/riders.

Some works have addressed using blockchain in ride sharing organization [124, 125, 126]. In [127], a general blockchain-based intelligent transportation framework is proposed. Also, a case study is presented to discuss the impact of using blockchain in real-time ride-sharing service. Semenko et al. [128] have proposed a distributed platform for ride-sharing service. The authors suggest having an overlay network that includes all ride-sharing agencies, called service nodes, to constitute the network layer. The service node is responsible for matching drivers' offers with riders' requests. However, the platform needs trusted infrastructure to run it, and hence it may fail in case that one service node is compromised. Meng et al. [129] have proposed a ride-sharing scheme using vehicular fog computing. Road Side Units (RSUs) installed at roads enable local matching between riders' requests and drivers' offers. Anonymous authentication is used to authenticate users by a trusted third party that can trace the real identities of malicious users. A private blockchain made of RSUs is presented to record ride-sharing data in an immutable ledger to enable data auditability. However, using limited resources devices, such as RSUs, to store massive records of ride-sharing data may be impractical especially in urban areas where there is a high demand of ride-sharing service.

Unfortunately, none of the existing research works have been proposed to decentralize ride-sharing services while ensuring privacy using blockchains. In this dissertation, we explore using the public blockchains to design a ride-sharing organization scheme while preserving users' privacy. We also discuss a security countermeasure against riders/drivers who try to make requests/offers while not committing to them.

2.3 Charging Coordination Schemes for Energy Storage Units

Technology advancements have been everywhere [130, 131, 132, 133]. Several works have investigated the problem of charging coordination in the smart grid, such as [134, 135, 136] and

privacy in different scenarios [137, 138, 139, 140, 141, 142, 143, 144]. In [134], a distributed vehicle-to-grid (V2G) control system is proposed to satisfy the EVs charging requirements. Tushar et. al. [135] propose an energy management technique to encourage EVs' owners to participate in energy trading using game theoretic approach. Sortomme et. al. [136] have proposed an algorithm to optimize energy and ancillary services scheduling. The algorithm maximizes profits to the EVs while providing additional system flexibility and peak load shaving to the utility and low costs of EV charging to the customer. However, these works do not take privacy preservation and security into account.

In [145], a privacy-aware charging coordination scheme is proposed. The idea is that each ESU sends a charging request to a local aggregator. The aggregator forwards the requests to the CC to run a charging coordination scheme to determine the ESUs with height priorities to charge them. Finally, to prevent the CC from linking the requests sent from an ESU, each ESU adds a noise to its sensitive data, such as TCC and SoC. The results indicate that increasing the level of noise increases the privacy level but degrades the charging coordination performance, and hence there is a trade-off between the charging coordination performance and privacy preservation.

In [92], the authors have proposed centralized and decentralized privacy-preserving and collusion-resistant charging coordination schemes for ESUs. The centralized charging coordination (CCC) scheme is used in case there is a robust communication infrastructure that connects the ESUs to the CC run by the utility, whereas the decentralized charging coordination (DCC) scheme is useful in case of remote areas and isolated microgrids in which a robust communication to the utility is unavailable and charging should be managed locally. In the CCC scheme, each ESU acquires anonymous tokens from the CC to authenticate its charging requests and send them to the CC via a local aggregator. Therefore, if the CC and the aggregator collude, they cannot identify the senders of the charging requests. To prevent the CC from being able to launch linkability attacks that aim to link charging requests data including TCC and SoC sent from the same ESU at con-

secutive time slots, an ESU needs to send multiple charging requests with random TCC and SoC values instead of only one request. After receiving the charging requests, the CC runs a technique to compute charging schedules that can maximize the amount of power delivered to the ESUs before the charging requests expire without exceeding the available charging capacity. In the DCC scheme, charging is coordinated in a distributed way using a privacy-preserving data aggregation technique so that each ESU selects some ESUs to act as proxies, and shares a secret mask with each proxy. Then, each ESU adds a mask to its charging request and encrypts it, so that by aggregating all requests, all masks are nullified and the total charging demand is known and used to compute the charging schedules.

Unfortunately, the existing research works do not present a charging coordination scheme that enables decentralized, transparent, and privacy-preserving charging coordination, which schedules charging requests in a manner that efficiently utilizes the power grid capacity while not stressing the grid.

2.4 Secure and Privacy-Preserving Energy Trading Schemes for Electric Vehicles

There are some research works that have been proposed to facilitate energy trading of the EVs using blockchain technology.

Recently, inspired by bitcoin, a PriWatt system is introduced by [146] that enables a blockchain-based private decentralized energy trading system. The system allows peer-to-peer energy trading without the need for a third-party intermediary. Pustisek et al. [74] use smart contracts to dynamically select the best bids from various charging stations. The idea is that EVs send charging requests with their planned routes, car battery status and driver preferences to the blockchain, and then the charging stations send their offered prices so that the EV choose the best price. Nevertheless, the proposed scheme does not consider charging reservation, or the underlying payment mechanism.

In [69], charging stations make bids to EV owners in response to their charging requests.

To select their preferred charging bids, EVs send hidden commitments to the blockchain. These commitments are used later so that the EV can prove to the charging station that it made the reservation. However, the charging station does not know if the charging point has been reserved or not until the EV drives to it and this may lead to poor management of the service by the CS. Furthermore, the paper does not propose an anonymous payment method to protect the privacy of the EVs. Also, it does not consider the Sybil attacks in which attackers may pretend as multiple non-existing EVs and launch severe attacks such as denial of service (DoS) attack by sending large number of reservation requests to block charging stations from getting customers to use the service.

Other schemes have used blockchain for energy trading but they focus on optimization techniques of the electricity pricing and the amount of traded electricity. Kang et al. [70] have proposed a consortium blockchain to establish a decentralized electricity trading system for V2V charging. To optimize electricity pricing and the amount of traded electricity among EVs, an optimization technique, called iterative double auction, has been used. In [75], the authors have introduced a permissioned energy blockchain system to implement secure charging services for EVs. The paper uses a Byzantine fault tolerance consensus algorithm to reach the consensus in the energy blockchain. The contract theory has been also used to satisfy the energy needed from EVs while maximizing the operator's profit. Huang et al. [76] have proposed an optimal charging scheduling algorithm that considers different vehicle charging scenarios such as V2V and CS2V. A double-objective optimization model has been used to maximize the user's satisfaction and minimizing users' cost.

In addition, an anonymous payment system is essential in any energy trading system and several schemes have been proposed in the literature. For example, Gunukula et al. [147] have proposed a dynamic charging scheme where an EV uses anonymous coins that are generated with partially blind signatures to prevent linking charging requests to a specific EV owner. Zhu et al. [148] also employ anonymous coupons that are issued by a third-party server, for the payment of parking fees. Another payment scheme is introduced by Au et al. [149] which, in addition to

anonymity, it implements a feature called voluntary revocation. That is, given the user's consent, it is possible to trace that user's transactions in case that his car was stolen. Nevertheless, all the aforementioned methods employ a trusted third-party to avoid double spending attacks, which is a single point of failure and a possible target for attackers.

In general, the existing works do not propose a comprehensive energy trading system that can achieve privacy of EVs' owners. Moreover, although anonymity is important to preserve the privacy of the owners of the EVs, some EVs may abuse this anonymity to launch Sybil attacks in which attackers may pretend as multiple non-existing EVs and launch severe attacks such as DoS attacks. Specifically, a malicious charging EV may launch a DoS attack by making many fake reservations to block other EVs from charging , and thus make the energy trading system unavailable. Moreover, an anonymous and secure payment system is needed so that the EV owners' privacy is protected while attacks such as double-spending and coin stolen are thwarted.

CHAPTER 3

INCENTIVIZED AND SECURE BLOCKCHAIN-BASED FIRMWARE UPDATE AND DISSEMINATION FOR AUTONOMOUS VEHICLES

In this chapter, we present our incentivized and secure blockchain-based firmware update and dissemination for autonomous vehicles

3.1 Introduction and Main Contributions

In this chapter, we propose an incentivized blockchain-based firmware update scheme tailored for AVs. We use blockchain and smart-contract technology to guarantee the authenticity and integrity of the updates. We also exploit the AVs' inter-communication capability to incetivize AVs to participate in the distribution of new firmware updates, therefore, ensuring high availability and fast delivery of the updates. Our main contributions are outlined as follows:

- A high availability and reliability is ensured by incentivizing AVs to participate in the distribution of the firmware updates. Distributor AVs are rewarded for their participation and the smart contract is used to manage the reward system and keep track of the reputation credit of each AV.

- Attribute-based encryption (ABE) technique is used to allow manufacturers to set a policy about who have the right to download and use an update. The access policy is defined on the smart contract that enforces its execution without an intermediary, so only authorized AVs can request and receive the update.

- A distributor can exchange an encrypted version of the update in return for a proof of distribution, i.e., a signature from a receiver AV. The smart contract delivers the key needed to

decrypt the update once the distributor sends the proof of distribution. The smart contract also increments the distributor's reputation score based on the received proofs.

- For efficient on-chain computation overhead, we use aggregate signature technique, where the distributor AVs can aggregate multiple proofs of the firmware update distribution into a single short signature, and then send it to the smart contract.
- A detailed proof-of-concept implementation using real devices to emulate both the blockchain nodes and the AVs. The results indicate that our scheme can ensure firmware update dissemination with low computation overhead.

The rest of this chapter is organized as follows. In Section 3.2, we discuss some preliminaries. Then, our proposed scheme is presented in details in Section 3.3. Detailed performance evaluations including our proof-of-concept implementations are provided in Section 3.4.

3.2 Preliminaries

In this section, we present the necessary cryptographic primitives that are used in our scheme. The notations used in this chapter are listed in Table 3.1.

3.2.1 Attribute Based Encryption

Attribute based encryption (ABE) is an encryption scheme that allows access control over encrypted data. In ABE, each user is assigned a set of secret keys corresponding to his/her set of attributes. Then, a message is encrypted under an access policy formed from a set of attributes. The ciphertext can only be decrypted by the users who have the attributes that can satisfy the policy. In our scheme, we use the attribute based encryption scheme proposed in [150] to enable a distributor AV to identify the neighbouring AVs who have the required features (attributes) to download a

Table 3.1: Notations used in this chapter.

Symbol	Description
$\mathcal{M}_\theta$	A manufacturer company for AVs.
PK_θ/SK_θ	Public/ Private key pair for manufacturer $\mathcal{M}_\theta$.
PK_{V_j}/SK_{V_j}	Public/ Private key pair for vehicle V_j.
$\mathcal{PK}_{U_i}/\mathcal{VK}_{U_i}$	zk-SNARK proving/verifying key pair for update (U_i).
U_i	ith firmware update version.
P_i	Access policy defined by the manufacturer for U_i.
AC_1	Authentication code of update U_i and policy (P_i).
V_j	A responder vehicle j that receives an update U_i.
k_j	Encryption Key for U_i of vehicle V_j.
h_i	Hash of k_j.
$\hat{U}_i$	The firmware update (U_i) encrypted with k_j.
$\mathcal{C}_{V_j}$	A concatenation of AC_i and h_j.
σ_j	A signature of receiving update U_i from vehicle V_j.

firmware update. This scheme is a ciphertext-policy attribute-based encryption (CP-ABE), where the access policy is embedded in the ciphertext.

3.2.2 Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK)

zk-SNARK [151] is a proof construction in which one, called the prover, can prove possession of a specific information, called a witness (w), e.g., a secret key, to someone else, called the verifier, without revealing that information. zk-SNARK does not require any interaction between the prover and verifier. We adopt the zk-SNARK scheme in [152]. Formally speaking, let L be an NP language with C as its decision circuit. Two keys play an essential role, namely, the proving key ($\mathcal{PK}$) and the verifying key ($\mathcal{VK}$). The proving key allows any prover to compute a proof π for a statement $y \in L$ with a witness w. Typically, a zk-SNARK scheme consists of the following three polynomial-time algorithms:

1. $Gen(1^{\lambda}, C) \rightarrow (\mathcal{PK}, \mathcal{VK})$. Given a security parameter λ and C as a decision circuit, the Gen algorithm generates two public keys, including $\mathcal{PK}$ and $\mathcal{VK}$, that are used to prove/verify the membership in L.

2. $Prove(\mathcal{PK}, y, w) \rightarrow \pi$. Given $\mathcal{PK}$, instance y, and witness for an NP statement w, the $Prove$ algorithm generates a proof π for the statement $x \in L_c$.

3. $Verify(\mathcal{VK}, y, \pi) \rightarrow \{0, 1\}$. Given $\mathcal{VK}$, instance y, and the proof π, the $Verify$ algorithm outputs 1 if $y \in L_c$, allowing the verifier to verify the instance y.

3.2.3 Aggregate Signatures

Given n signatures $(\sigma_1, \ldots, \sigma_n)$ on n distinct messages from n users, aggregate signature scheme can be used to aggregate all these signatures into a single short signature (σ_{agg}) where the size of the aggregated signature is similar to the size of the individual signatures. Then, given the users' public keys, σ_{agg} and the n messages, a verifier can efficiently ascertain that the n users indeed signed the messages where it is much more efficient to verify the aggregated signature than the individual signatures. In our scheme, we use the aggregate signature scheme proposed in [153] to reduce the on-chain computation overhead. The idea is that by using aggregate signature technique, a distributor AV can aggregate multiple proofs, i.e., signatures of the firmware update distribution into a single short signature, and then send it to the smart contract.

3.3 Proposed scheme

In this section, we present our scheme that aims to ensure secure and salable delivery of firmware updates from automobile manufacturers to AVs. We first present a general architecture for the scheme, followed by system initialization, the smart contract creation, firmware update dissemination, and rewarding.

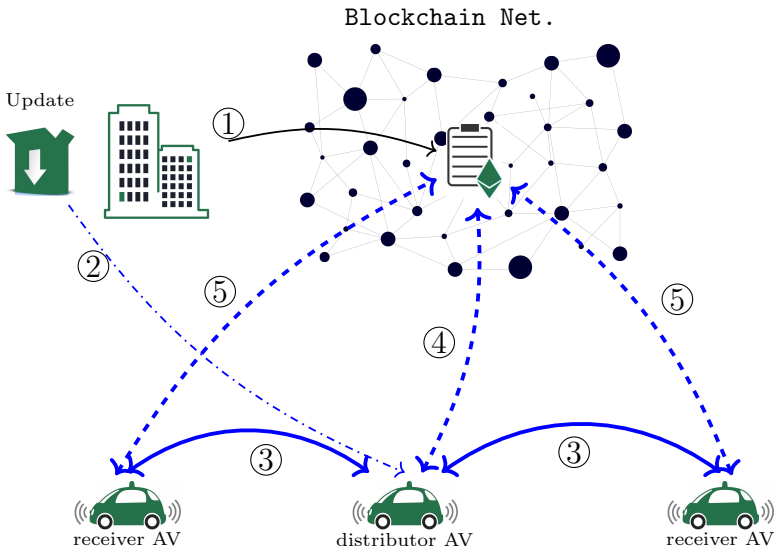


Figure 3.1: The exchanged messages in our schemes: (1) The manufacturer creates a smart-contract for a new firmware update by including its hash code for authenticity checking by AVs. (2) The manufacturer sends the new update to top-reputation AVs (distributors). (3) A distributor exchanges an encrypted version of the update in return for proof of reception of the update by a responder AV. (4) A redeem transaction, containing multiple proofs, is sent to the smart contract to update the distributor's reputation. (5) The responder AV receives the decryption key of the firmware update from the smart contract.

3.3.1 Network Architecture

Fig. 3.1 shows the network architecture, which is comprised of two manufacturers with their AVs, two smart contracts for firmware updates of each manufacturer, and a consortium blockchain.

An illustration to the exchanged messages in our scheme is shown in Fig. 3.1, and more details are shown in Fig. 3.2. The role of each entity is discussed in the following paragraphs.

Manufacturer. The manufacturer is responsible for keeping its AVs updated with the latest versions of the different firmware of the subsystems of the AVs. During the manufacturing process of AVs, the manufacturer uploads each AV with a set of cryptographic keys and public parameters that will be used to ensure secure distribution of firmware updates. Also, each time a new update is released, a corresponding smart contract is deployed by the manufacturer to allow AVs to check the

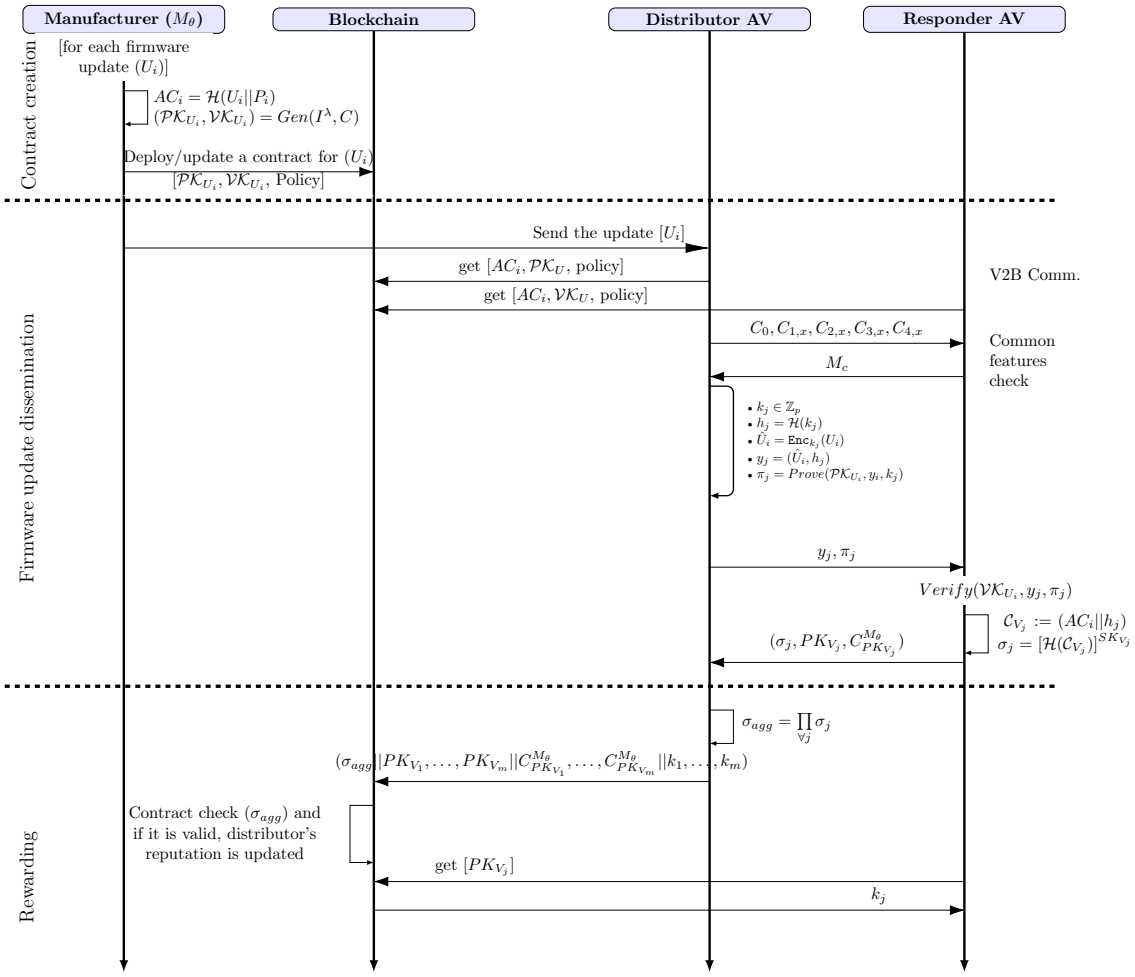


Figure 3.2: Content of exchanged messages in our scheme.

integrity and authenticity of the update. In addition, to attract AVs to participate in the distribution of an update, the manufacturer compensates the participants through a rewarding mechanism, e.g., momentary rewards and free or reduced-price maintenance services.

Autonomous Vehicles. We distinguish between two types of AVs, *distributors* and *responders*. The distributor AV disseminates a new firmware update to other AVs (responders) in its vicinity. Each responder AV that receives an update can also act as a distributor of that update. By this way,

we can ensure the large-scale dissemination of the update quickly. Initial distributors are selected by the manufacturer based on their reputation scores which are recorded in a smart-contract.

Smart Contract. For each new firmware update, a smart contract is created. The contract contains the necessary credentials allowing any receiver of the update to authenticate it and verify its integrity. In addition, the contract implements the reputation logic that evaluates and keeps track of the AVs' activities in the distribution of the firmware update. More specifically, the contract increases the reputation score of a distributor AV after receiving proofs of the firmware distributions from responder AVs.

Blockchain Network. Blockchain network is at the center of our scheme and it is responsible for the execution of the smart contracts in a distributed manner without relying on a single trusted central party [154, 155, 156]. This is mandatory to ensure a scalable and secure firmware update dissemination. Moreover, to improve the efficiency of the scheme, we opt for a *consortium blockchain*, where the validators, i.e., nodes that have permission to write on the shared ledger, are known and trusted entities. In our case, the validators can be the manufacturers of the different automobile brands.

3.3.2 Initialization

In our scheme, we use the attribute based encryption scheme proposed in [150]. The scheme is based on multi-authority attributes, where each manufacturer is considered as an authority that decides and distributes a set of attributes (or features) for its AVs. $\mathbb{M}$ is the set of all available manufacturers and a manufacturer $\mathcal{M}_\theta \in \mathbb{M}$. Let $\mathbb{A}$ be the set of all attributes (or features), an access policy (A, δ) on $\mathbb{A}$ with $A \in \mathbb{Z}_p^{l \times n}$, called the share generating matrix in the field $\mathbb{Z}_p$ of prime order p with l rows and n columns, and a function δ that labels the rows of A with attributes from $\mathbb{A}$, i.e., $\delta: [l] \rightarrow \mathbb{A}$. In addition, let ρ be a function that maps attributes in rows to its manufacturers,

where $\rho: [l] \rightarrow \mathcal{M}_\theta$. Consider e: $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow G_T$ a cryptographic bilinear map with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are multiplicative group. Each manufacturer $\mathcal{M}_\theta \in \mathbb{M}$ should select two random elements $(\alpha_\theta, y_\theta) \overset{R}{\leftarrow} \mathbb{Z}_p^*$ as its secret keys, and then, $\mathcal{M}_\theta$ can compute its public key as $PK_\theta = \{e(g_1, g_2)^{\alpha_\theta}, g_1^{y_\theta}\}$. Besides, a public hash function $\mathcal{H}: \{0,1\}^* \rightarrow G_1$ is used to map an AV global identifier GID to a point in G_1, public hash function $F: \{0,1\}^* \rightarrow G_1$ that maps an attribute $a \in \mathbb{A}$ to G_1, and a function T that maps an attribute $a \in \mathbb{A}$ to the manufacturer $\mathcal{M}_\theta$, hence, the function $\rho(\cdot)$ can be redefined as $\rho(\cdot)$: $\mathrm{T}(\delta(\cdot))$. The global parameters are then defined as $GP = \{\mathbb{G}_1, \mathbb{G}_2, G_T, \mathbb{Z}_p, \mathcal{H}, F, T, \mathbb{A}, \mathbb{M}\}$.

Besides, during the production, $\mathcal{M}_\theta$ should assign each AV a key for each assigned attribute $a \in \mathbb{A}$ using the AV global identity GID as follows: $\mathcal{M}_\theta$ chooses a random $t \overset{R}{\leftarrow}, \mathbb{Z}_p^*$ and outputs to the AV attributes secret keys as

$$SK_{GID,a} = \{K_{GID,a} = g_2^{\alpha_\theta} H(GID)^{y_\theta} F(a)^t, K'_{GID,a} = g_1^t\}.$$

Finally, for each AV, $\mathcal{M}_\theta$ generates a public/private key pair as follows: a random element $x_a \overset{R}{\leftarrow} \mathbb{Z}_P$ is selected as the private key and the corresponding public key is $PK_{V_j} = g_2^{x_a}$. The manufacturers should obtain a certificate ($C_{PK_{V_j}}^{M_\theta}$) for its public key from the certificate authority. Then, the public/private key pair (PK_{V_j}, x_a) and $C_{PK_{V_j}}^{M_\theta}$ are added to the AV's tamper proof device along with manufacturer's public key PK_θ.

3.3.3 Smart Contract Creation

Upon releasing a new update by a subsystem's manufacturer, denoted by U_i, the AV manufacturer that uses the subsystem should first test the update. Note that a subsystem's manufacturer may be different from the AV manufacturer. If the AV manufacturer decides to use it on its AVs, it starts the firmware update as follows. The manufacturer creates a smart contract and initializes it by two attributes: (1) A proving/verifying key pair: $(\mathcal{PK}_{U_i}, VK_{U_i}) = Gen(1^\lambda, C)$, required for

the execution of the zk-SNARK protocol; (2) An authentication code for the new firmware update: $AC_i = \mathcal{H}(U_i||P_i)$, where P_i is the access policy defined by the manufacturer to deliver the update to only AVs that have the features defined in the policy.

The manufacturer deploys a smart-contract by broadcasting a transaction to the blockchain network. The deployed smart-contract is described in Algorithm 3.1 and includes the following main functions:

- *Authenticity and integrity of a firmware update.* Since the update's authentication code and verification key are stored in the contract, AVs, by the help of the blockchain, can check whether a received firmware update is the same one that was originally approved by the AV manufacturer.

- *AVs' reputation.* When an AV participates in the distribution of a new update, the proof of distribution is sent to the smart contract which in turn increases its reputation score. The manufacturer rewards the highly-reputed AVs, i.e., the active AVs in distributing the firmware. The reward can be momentary, free or reduced-price maintenance service, etc.

- *Firmware access control.* Each firmware update has an access policy set by the manufacturer, and the AVs that have enough features to satisfy the policy can receive the firmware. This can restrict the distribution of the firmware to only certain AVs. The access policy of an update is included in the update's smart-contract.

3.3.4 Firmware Update Dissemination

In this stage, the manufacturer starts the dissemination process of a new update by first selecting the most active AVs in distributing updates (based on their reputation scores) to act as the initial distributors. As discussed before, rewards are used to incentivize the AVs to act as distributors and actively distribute the new firmware, but the rewarding system should be secure to

Algorithm 3.1: Pseudocode for the Firmware Update Contract

```
1  contract FirmwareUpdate
2      mapping(address => int) Reputation // Mapping for distributors reputation
3      mapping(address => int) UpdatedAVs // Mapping for AVs with the No.of obtained
           updates
4      function FirmwareUpdate(_PK, _VK, _AC_i, _P_i, X)
5          PK ← _PK // Proving Key
6          VK ← _VK // Verifying key
7          AC_i ← _AC_i // authentication code
8          P_i ← _P_i // ABE Policy
9          MaxUpdate ← X // Max. No. of download per Update
10     function RecieveProof(σ_agg, PK[], C[] keys[])
11         address [] RecievedAVs // Received AV list
12         for s ← 0 to PK.lengh do
13             if verifySig(pk_M, PK[s], C[s])
14                 return
15             end
16             if UpdatedAVs[PK[s]] > MaxUpdate
17                 return
18             end
19             h_s ← H(keys[s])
20             C_{V_s} ← H(AC_i, h_s) RecievedAVs.push(Pairing(PK[s], C_{V_s}))
21         end
22         if Pairing(g_1, σ_agg)=Prod(RecievedAVs)
23             UpdateReputation(msg.sender, PK.length)
24             for i ← 0 to PK.lengh do
25                 emitEvent("KeyRevealed", PK_i, keys[i])
26                 UpdatedAVs[PK_i]← UpdatedAVs[PK_i]+1
27             end
28         end
29     function UpdateReputation(Dist, N)
           // increase reputation distributors
30         Reputation[Dist]← Reputation[Dist]+=N
```

ensure that only honest distributors which distribute the firmware are rewarded. It is anticipated that each AV manufacturer will use many subsystems made by different companies, and therefore, it is very likely that different AV manufacturers may use the same subsystem produced by the same company in their vehicles. Thus, it is very important for each manufacturer to ensure that its distributors will deliver a particular update to only certain models of its AVs.

Thanks to ABE, as presented in Section 3.2, each manufacturer can define an access policy for each update on the associated smart contract, where only AVs that belong to the same manufacturer

and have enough features, such as model, year of manufacturer, etc, can decrypt and use the firmware update they got from a distributor.

For a distributor to find other AVs which can satisfy the access policy of an update and deliver it, the following steps should be taken.

1. A distributor AV first queries the blockchain for the AC_i, proving key $\mathcal{PK}_{U_i}$, and the manufacture's access policy (A, δ).

2. Then, distributor AV should broadcast an encrypted challenge message (M_c) using the ABE to the nearby AVs. This message is encrypted using the manufacture's public key PK_θ under the access policy (A, δ) set by the manufacturer. Hence, only the AVs which owns the set of attributes that satisfy the policy are able to decrypt the ciphertext CT. To encrypt M_c, distributor AV first creates two random vectors $v = (z, v_2, \cdots v_n)^T$ and $w = (0, w_2, \cdots w_n)^T$, where $\{z, v_2, \cdots v_n, w_2, \cdots w_n\}$ are elements randomly selected from Z_p^*. We denote λ_x as the share of the random secret z corresponding to row x, i.e., $\lambda_x = (A_x \cdot v)$ and w_x denotes the share of zero, i.e., $w_x = (A_x \cdot w)$, where A_x is the x-th row of access matrix A. The distributor AV chooses a random element $t_x \xleftarrow{R} \mathbb{Z}_p^*$ for each row in the policy matrix A and computes the CT as:

$$
\begin{aligned}
C_0 &= M_c \cdot e(g, g)^z; \\
\{C_{1,x} &= e(g_1, g_2)^{\lambda_x} e(g_1, g_2)^{\alpha_{\rho(x)} t_x}; \\
C_{2,x} &= g_1^{-t_x}; \\
C_{3,x} &= g_1^{y_{\rho(x)} t_x + w_x}; \\
C_{4,x} &= F(\delta(x))^{t_x}\}_{x \in [l]}
\end{aligned}
$$

3. After a responder AV receives CT, it first queries the smart contract for the access policy (A, δ), $\mathcal{VK}_{U_i}$ and AC_i. Then, to decrypt M_c, the AV should use the policy (A, δ) from the

blockchain and its secret keys $(K_{GID,a}, K'_{GID,a})$ for the subset of rows A_x of satisfied attributes and for each row x to compute

$$C_{1,x} \cdot e(K_{GID,\delta(x)}, C_{2,x}) \cdot e(H(GID), C_{3,x}) \cdot e(K'_{GID,\delta(x)}, C_{4,x})$$

$$= e(g_1, g_2)^{\lambda_x} \cdot e(H(GID), g_2)^{w_x}$$

Then, AV calculates the constants $c_x \in \mathbb{Z}_p^*$ such that $\Sigma_x c_x A_x = (1, 0, \ldots 0)$ and computes:

$$\Pi_x (e(g,g)^{\lambda_x} . e(H(GID), g)^{w_x})^{c_x} = e(g,g)^z$$

This is true because $\lambda_x = (A_x \cdot v)$ and $w_x = (A_x \cdot w)$, where $\langle (1, 0, \cdots, 0) \cdot v \rangle = z$ and $\langle (1, 0, \cdots, 0) . w \rangle = 0$. Hence, the challenge message can be decrypted as $M_c = C_0 / e(g,g)^z$.

4. Finally, once a responder AV manages to get M_c, the AV sends it to the distributor AV. Henceforth, both the distributor and responder AV can proceed with the firmware update transfer.

A distributor AV sends the firmware update to a responder AV in return for a signature, i.e., a proof for disseminating the firmware. This exchange of firmware update and proof can be made in a trust-less way using zk-SNARK protocol as follows.

1. The distributor AV generates a secret key $k_j \in \mathbb{Z}_p^*$ and calculates $h_j = \mathcal{H}(k_j)$.

2. Then, it computes $\hat{U}_i = \texttt{Enc}_{k_j}(U_i)$, where `Enc` is a symmetric-key encryption algorithm.

3. For zk-SNARK protocol, the secret witness is the instance $y_j = (\hat{U}_i, h_j)$ and k_j. The NP statement is as follows:

$$\exists\, k_j : \mathcal{H}(k_j) = h_j \ \wedge\ \mathcal{H}(Dec_{k_j}(\hat{U}_i), P_i) = AC_i, \tag{3.1}$$

which attests that the distributor AV has a key, k_j, such that its hash is h_j, and if k_j is used to decrypt $\hat{U}_i$, it will match the update authentication code AC_i. After that, the distributor computes a zero-knowledge proof $\pi_j = Prove(\mathcal{PK}_{U_i}, y_j, h_j)$ and sends $y_j||\pi_j$ to the responder.

4. Upon receiving $(y_j||\pi_j)$, the responder first verifies that $Verify(\mathcal{VK}_{U_i}, y_j, \pi_j) = 1$, and then computes a signature $\sigma_j = [\mathcal{H}(\mathcal{C}_{V_j})]^{SK_{V_j}}$, where SK_{V_j} is its private key and $\mathcal{C}_{V_k} = (AC_i, h_j)$.

5. Finally, it sends $(\sigma_j||PK_{V_j}||C^{M_\theta}_{PK_{V_j}})$ to the distributor.

3.3.5 Rewarding

In this phase, the distributor AV sends a redeem transaction, containing multiple proofs, to the smart contact to update its reputation score proportionally to the number of AVs which received the update. This rewarding process is done as follows.

1. To reduce the number of transactions sent to the blockchain, instead of making a transaction each time a firmware is transferred, one transaction can be sent for several firmware transfers efficiently, as follows. Once the distributor AV gets the proofs of transferring an update (σ_j) from other vehicles, it aggregates multiple signatures into a single signature (σ_{agg}) as follows: $\sigma_{agg} = \prod_{\forall j} \sigma_j$.

2. The distributor sends a transaction to the blockchain containing

$$\sigma_{agg}||PK_{V_1}, \ldots, PK_{V_m}||C^{M_\theta}_{PK_{V_1}}, \ldots, C^{M_\theta}_{PK_{V_m}}||k_1, \ldots, k_m$$

where m is the number of vehicles that received the update.

3. The smart contract method `RecieveProof` first verifies that the received public key is one of the certified keys by the manufacturer (see `verifysign` in Algorithm 3.1) as well as many number of times that AV gets the update. Then, it computes $h_j = \mathcal{H}(k_j)$ and $\mathcal{C}_{V_j} = \mathcal{H}(AC_i, h_j)$ for all j. Thereafter, it verifies the aggregated signature by checking if $e(g_1, \sigma_{agg}) =$

$\prod_{\forall j \in m} e(PK_{V_j}, \mathcal{C}_{V_j})$ or not (see `pairing` in Algorithm 3.1 which can be executed by a pre-compiled contract for elliptic curve pairing operations available in [157]).

4. Finally, the distributor is rewarded by increasing its reputation score proportionally to the number of vehicles that received the update (see the method `UpdateReputation` in Algorithm 3.1).

5. A responder AV queries the contract for a relevant event associated with its public key for the decryption key k_j it needs to decrypt $\hat{U}_i$ to get U_i (see event `"KeyRevealed"` in Algorithm 3.1).

3.4 Performance Evaluations

In this section, we evaluate our proposed scheme.

3.4.1 On-chain Overhead

Methodology/Experiment setup. We have implemented a private blockchain that is run using instances of Geth inside of Docker containers, built on a modification of the code seen in [158]. Geth is an implementation of Ethereum blockchain in Go, and allows us to set up nodes (including validators/miners) with our blockchain. We have set-up several Docker containers running on a workstation machine acting as blockchain nodes, i.e., miners/validators. To emulate AV hardware, we used Raspberry Pi devices. Table 3.2 gives the specifications for the workstation and the Raspberry Pi devices used in our implementations.

Performance metrics. We define the following key performance metrics for evaluating the on-chain overhead in our proposed scheme.

- *On-chain execution time.* The time elapsed between sending a transaction by an AV and returning a transaction hash by the blockchain indicating that the transaction is successful.

Table 3.2: Computer specifications used in our experiments.

	PC	Raspberry Pi
CPU	2.00GHz 8-core Intel i7-4765T	1.2GHz 4-core ARM Cortex-A53
Memory	7.7GB	1GB
Operating System	Ubuntu 18.04.2	Raspbian 9.9

- *Total time for the firmware update:* This time includes the overall time for delivering the new update to an AV. Specifically, it includes the off-chain computation overhead for firmware distribution and the on-chain computation time for the BLS verification time, reputation update, and key retrieval.

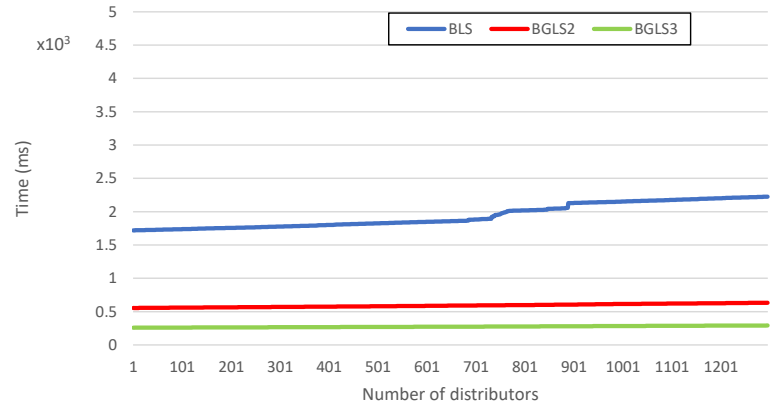
  We measured the on-chain execution time by measuring the total time elapsed between sending a series of transactions from an AV and returning a transaction hash by the blockchain network indicating that the final transaction was successful. The specific transactions in this case are a call to a function which takes up to 3 signatures (aggregated or individual), messages, and elliptic curve point pairs before sending the information to be verified using the appropriate verification algorithm (BLS or BGLS). If the verification is successful, the sender's reputation score is updated appropriately, and the decryption key is revealed to the AV.

- *Storage cost overhead.* This is the cost of storing the data in blockchain. According to the Ethereum Yellow paper [2], storage cost is calculated by measuring the gas cost of transactions as follows:

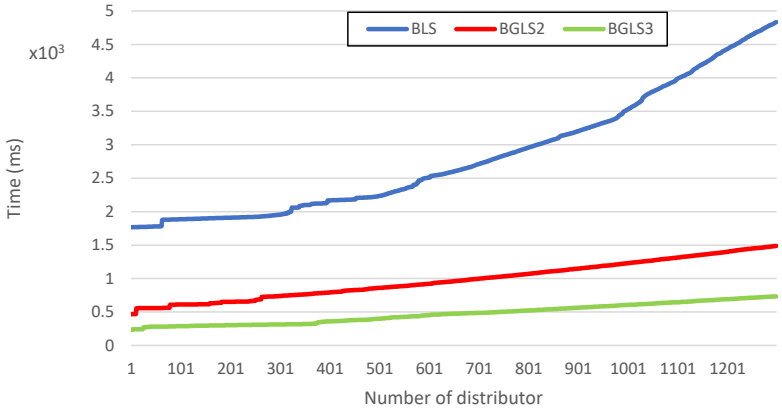
$$\text{Storage cost} = \frac{GasCost - 21000}{68} \tag{3.2}$$

Results and Discussion. Using the above key metrics, wen have evaluated the performance of our scheme. The experiments were executed using a program written in Node.js [159].

The first two experiments are related to the BLS/BGLS verification time with two and six validators, and their results are shown in Fig. 3.3. In these experiments, 1300 transactions were sent



(a) With two validators.



(b) With six validators.

Figure 3.3: Execution time of verifying proofs of distribution.

to the blockchain (simulating 1300 distributor AVs), with each being timed individually. Fig. 3.3 shows the the execution time of validating proofs sent by distributors. It can be clearly seen that in case of using BLS signatures: as the number of distributors increases, the time taken by the validators increases to reach consensus. It can also be seen that in the case of using BGLS, the execution time is lower than that of BLS. Moreover, as the number of distributors increases, there is no significant increase in the execution time. This is because in BGLS, the validators only verify a batch of proofs at once, while in BLS, the validators should verify proofs individually.

In another part of the experiment, we increased the number of validators to asses the impact

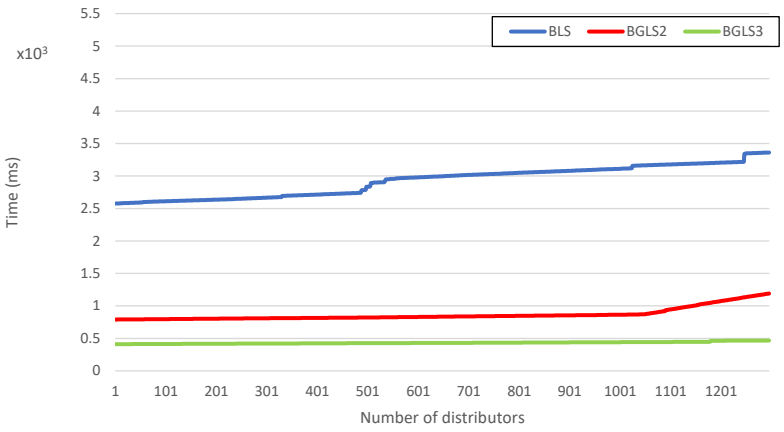
on execution time. The results indicate that the exection time is lower in case of two validators because:

- Having more validators increases the amount of time for the validators to reach a consensus before a block is sealed and added to the blockchain.
- Since the validators are run inside docker containers on a single machine, the available resources are greater per validator when only two validators are in use. Comparatively, the same resources are divided among six validators, resulting in less overall computation power per validator.

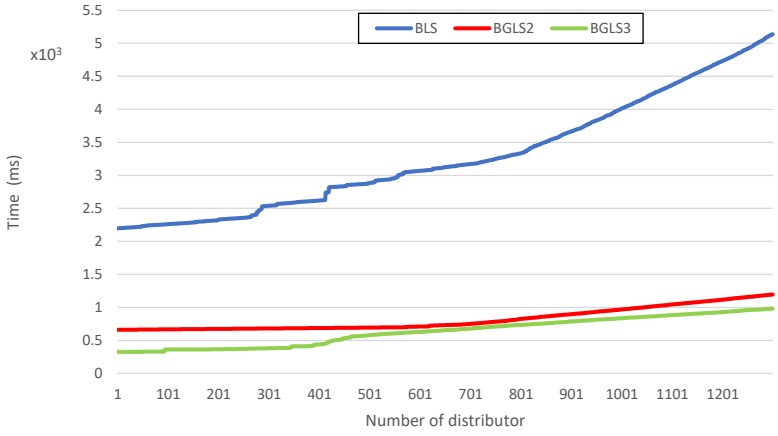
We also run experiments to measure the total execution time for signature verification, reputation update, and key retrieval. Similar to the previous experiments, 1300 transactions were sent to blockchain simulating 1300 distributors with each being timed individually. Fig. 3.4 shows the results of these experiments. It can be seen that the total execution time increases as more distributors interact with the contract. Also, the execution time is greater for six validators compared to two validators. It can be concluded that the use of BGLS is efficient since the time needed for updating the AVs is very short comparing to the BLS.

Finally, we measured the storage overhead needed by our scheme. Typically, transactions on Ethereum blockchain have a fixed gas cost measured in Gwei (1 ether = 10^{18} Gwei), which means that we can accurately predict the gas cost for large amounts of transactions. The following gas costs correspond to each verification type:

- *BLS:* 40237 Gwei
- *BGLS with 2 signers:* 61,016 Gwei
- *BGLS with 3 signers:* 67,866 Gwei



(a) with two validators.



(b) with six validators.

Figure 3.4: Execution time for verification of the proofs, reputation score update, and key reveal.

Eq. 3.2 is used to convert the gas cost into total storage overhead. Fig. 3.5 shows the number of bytes used for BLS and BGLS verification for 25, 50, 100, and 200 distributors. BLS consistently requires more storage space, as a new function call must be made for each verification, whereas BGLS can perform multiple verifications with a single function call.

3.4.2 Off-chain Overhead

In this section, we evaluate the computation overhead of the cryptographic operations used in our scheme.

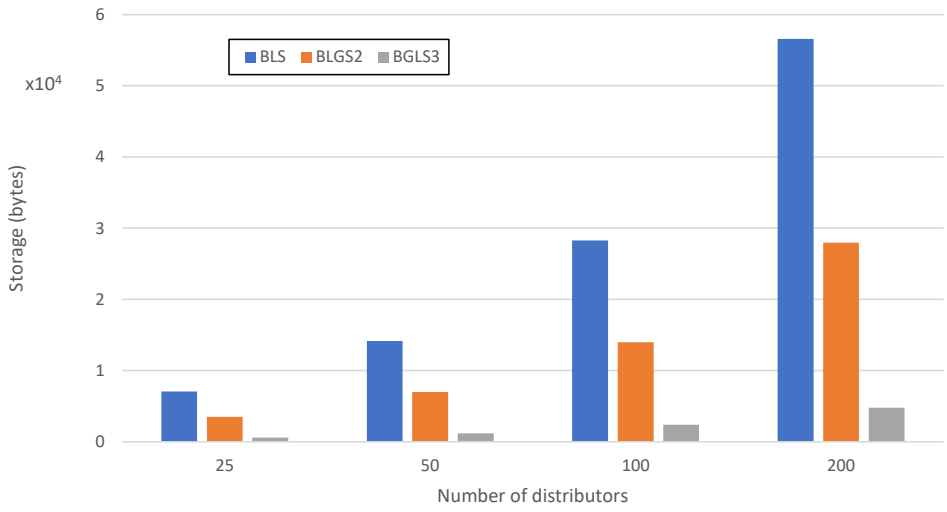


Figure 3.5: Storage overhead comparison.

The computation times of ABE are measured using Intel Core i7- 4765T 2.00 GHz and 8GB RAM machine and Python charm cryptographic library in [160]. The off-chain computation overhead on each entity is summarized in Table 3.3. In our scheme, a distributor AV needs to broadcast a challenge message (M_c) encrypted by a number of attributes (γ) specified in the smart contract. According to [160], the required time for encryption is ($10.9\times\gamma + 1.35$) ms. In addition, a responder AV needs to decrypt the ciphertext with total decryption time of ($4.03\times\gamma + 0.01$) ms. After running the ABE, zk-SNARK protocol should be run. In this protocol, a proof is generated by the distributor and then verified by the responder. We have implemented the NP statement in Eq. 3.1 using Zokrates[1] toolbox. MIMC [161] is used for encryption/decryption due to its efficiency with zk-SNARK proofs, and SHA256 is used for hashing. The time needed to generate proof is 6 seconds, where, the verification of the proof is 5 milliseconds. It should be noted that the distributor AV can generate multiple proofs offline before starting the communication session with the responder AVs. Hence, the total computation time needed to run ABE and zk-SNARK verification of our scheme is low, which is suitable for our application because the AVs are in motion and they can

[1] https://github.com/Zokrates/ZoKrates

Table 3.3: Off-chain computation overhead. α is the number of attributes of the underlying ABE.

Entity	Operation	Computation Overhead
Distributor	ABE Encryption	$(10.9 \times \alpha + 1.35)$ ms
	Proof Generation for ZKSNARK	$6s$
Responder	Decrypting the challenge packet	$(4.03 \times \alpha + 0.01)$ ms
	Verifying ZKSNARK	5 ms
	Composing a response packet	40 ms

only communicate in a short time.

In our scheme, blockchain is required to ensure the authenticity and integrity of the new update. To reduce the cost needed to execute our scheme on the blockchain, most of the computations needed to secure the scheme are done outside the blockchain. Additionally, our scheme reduces the on-chain operations by reducing the number of transactions sent to the blockchain by aggregating several firmware transfers in one transaction using aggregate signature scheme. Also, as discussed before, the computation overhead to run our scheme is low. For the actual time of exchanging the update, according to [13], the mean throughput for delivering data to moving vehicles that use IEEE 802.11 protocol is equal to 760 kbit/s. If we assume that the size of a firmware update is 1 MByte, then the time required to transfer the update is 1.3 seconds. Therefore, given this transfer time and the short time needed for the cryptographic computations, our scheme can be executed during the contact time of two moving AVs.

CHAPTER 4

RIDE SHARING ORGANIZATION WITH PRIVACY-PRESERVATION, TRUST AND FAIR PAYMENT ATOP PUBLIC BLOCKCHAIN

In this chapter, we present our privacy-preserving blockchain-based ride sharing organisation scheme.

4.1 Overview and Main Contributions

In this chapter, we propose a **B**lockchain-based **Ride** sharing organization scheme, called B-Ride, that is privacy-preserving and establishes trust between drivers[1] and riders. *To the best of our knowledge, this work is the first to propose ride-sharing organization scheme atop open and permissionless blockchains.* Table 4.1 summarizes the main differences between our proposed scheme and existing works. Our main contributions and the challenges we aim to address can be summarized as follows:

1. A blockchain-based scheme is proposed to realize decentralized ride-sharing service. To preserve riders' privacy, we use location *cloaking* technique, where a rider posts cloaked pick-up and drop-off locations as well as pick-up time. Then, interested drivers use off-line matching technique to check if a request's location falls on his route and then send the exact trip data encrypted with rider's public key. Then, a rider can select the best-matched driver to share a trip. This acts as a distributed auction that is handled through the blockchain to ensure transparency and security.

[1] Hereafter, we use the term driver to refer to individuals or companies that own vehicles or buses that can be used in ride-sharing service.

Table 4.1: Comparison between B-Ride and existing RSS platforms

	Architecture	Privacy	Trust	Fair payment	Transparency
Current RSS [2]	Centralized	×	×	×	×
SRide: [113]	Centralized	√	×	×	×
Co-utile [39]	Decentralized	√	×	×	×
DACSEE [111], Arcade City [112]	Blockchain	×	×	×	√
B-Ride	Blockchain	√	√	√	√

Note: √ denotes a functionality a realized feature and × denotes an unrealized feature.

2. To boost trust between riders and drivers, we propose a time-locked deposit protocol for ride-sharing service based on the zero-knowledge set membership [162]. The idea is to define a *claim-or-fine* methodology that works as follows. (*i*) A rider has to post a smart contract with a deposit as a proof of accepting a driver's offer. (*ii*) The selected driver should also send a deposit to the contract as a commitment to his offer. (*iii*) Upon arrival at the pick-up location, the driver acts as (*a prover*) and sends a *proof for his arrival to pick-up location* to the blockchain. Specifically, the driver proves that the pick up location falls in a predefined set of cells. (*iv*) Finally, a smart contract acts as a (*verifier*) that checks the proof in a zero-knowledge manner, and then assigns a reward to driver in case of valid proof or a fine in case of invalid proof or if no proof is sent before the agreed pick-up time.

3. Also, we develop a method to ensure fair payment in a trust-less manner between the driver and rider. A driver needs to periodically send at each regular interval the elapsed distance to the rider who authenticates it by signing it using his private key. Then, once the rider provides a *proof-of-elapsed-distance* (i.e., the elapsed distance and driver's signature on it), the smart contract transfers the fare to the driver. In this way, the driver gets paid as he/she drives. Meanwhile, if the rider stops sending proofs to the blockchain, he/she can stop the trip immediately. Moreover, only elapsed distances are stored on the blockchain and no sensitive information such as exact locations are leaked to the public.

4. Finally, B-Ride computes the *reputation scores* of drivers based on their prior behaviours. Unlike, current centralized reputation approaches [163], our reputation management mechanism is decentralized using blockchain that is executed in a self-enforcing manner once a predefined set of conditions are met. Specifically, in B-Ride, each driver has two reputation scores; (*i*) The first score increases every time a driver sends a valid proof of his arrival to the pick-up location. (*ii*) The second score increases upon the completion of each trip. Based on the two scores, each driver will have a trust value in B-Ride that is used by riders to select good drivers for their trips. Our reputation mechanism makes economic incentive for drivers to behave properly, otherwise they will not be selected by riders.

5. To evaluate B-Ride, we have implemented it on top of Ethereum, a real-world public blockchain platform. Intensive experiments and performance evaluations are conducted in an Ethereum test network.

The rest of the chapter is organized as follows. In Section 4.2, we discuss cryptographic primitives used by B-Ride. We discuss the network and threat models, followed by the design goals of our scheme in Section 4.3. A detailed description of our scheme is presented in Section 4.4. Performance evaluations are presented in Section 4.5. We present the security/privacy analysis and performance evaluations of our scheme in Section 4.6.

4.2 Preliminaries

In this section, we present the necessary cryptographic primitives that are used in this chapter. The main notations used in this chapter are given in Table 4.2.

Table 4.2: Key notations in B-Ride.

Notation	Represent for
d, r	Denotes a driver or rider.
$\ell_k^{(d)}, t_k^{(d)}$	A pair of exact location and time.
$\Gamma^{(r)} = \left\{ \left(l_0^{(r)}, t_0^{(r)} \right), l_1^{(r)} \right\}$	A ride desired trip.
$\Gamma^{(d)} = \left\{ \left(l_0^{(d)}, t_0^{(d)} \right), \ldots, \left(l_\phi^{(d)}, t_\phi^{(d)} \right) \right\}$	A ride planned trip.
$\widehat{\Lambda^{(r)}} = \{C_o^{(r)}, C_d^{(r)}, T_o^{(r)}\}$	A rider r generalized pickup, drop off, and time.
$\delta^{(r)}$	Spatial slack distance that a rider willing to walk to driver pick up location.
$\tau^{(r)}$	Temporal slack delay time that a rider willing to wait.
$\mathcal{C}_{d_i}$	Encrypted ride-offer of a driver d_i.
β_d^{AP}	Driver' reputation score of arrival to pick-up. location
β_d^{AD}	Driver' reputation score of arrival to drop-off location.
$\mathcal{H}$	Secure hash function.
q, G_1, G_2, P, e	Public parameters of bilinear pairing

4.2.1 Bilinear Pairing

Let PG be a pairing group generator that on input 1^k outputs descriptions of multiplicative groups $\mathbb{G}_1$ and $\mathbb{G}_\mathrm{T}$ of prime order p where $|p| = k$. There exists an admissible bilinear map e:$\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_\mathrm{T}$ meaning that (1) for all $a, b \in \mathbb{Z}_p$, it holds that $e\left(g^a, g^b\right) = e(g,g)^{ab}$; (2) $e(g,g) \neq 1$; and (3) the bilinear map is efficiently computable. $\mathcal{H}$ is a collision-resistant hash function that maps a string of arbitrary length to $\mathbb{Z}_p$. We denote $u \in_R \mathbb{Z}_p$ as a randomly choosing a element from $\mathbb{Z}_p$.

4.2.2 Zero Knowledge Set Membership Proof (ZKSM)

A set membership proof enables a prover to prove, in a zero-knowledge way, that a secret value lies in a given public set without revealing the value. The set can perhaps be a list of cities or clubs. Typically, such proofs can be used, for example, in the context of electronic voting, where a voter is required to prove that his secret vote belongs to the set that contains all possible candidates.

We will use Camenisch and Stadler [162] notation for proofs of knowledge:

$$\boldsymbol{PK}(\delta, \gamma) : Y = g^{\delta} h^{\gamma} \wedge (\gamma \in \phi) \tag{4.1}$$

Where $Y = g^{\delta} h^{\gamma}$ is a Pedersen commitment of the integer $\delta \in \mathbb{Z}_p$ using a random number γ. The proof in Eq. 4.1 convinces the verifier that the secret in the commitment Y lies in the set ϕ without having to explicitly list ϕ in the proof. The set ϕ can be a common input to both prover and verifier. The set membership proof can be instantiated in the discrete logarithm setting and made non-interactive with Fiat-Shamir heuristic.

The security guarantees are:

- *Soundness.* No prover can convince an honest verifier if he/she did not compute the results correctly.
- *Completeness.* If the statement is true, the honest verifier (the one following the protocol properly) will be convinced of this fact by an honest prover.
- *Zero-knowledge.* The verifier can not learn anything other than the fact that the statement is true.

4.3 Network/Threat Models and Design Goals

In this section, we describe the considered network and threat models. Also, we define the threat and adversary assumptions.

4.3.1 Network Model

As depicted in Fig. 4.1, the considered network model has the following entities.

1. *Blockchain.* At the heart of our scheme is the blockchain network that handles all the ride-sharing transactions. We opt for a permissionless blockchain where everyone can use the

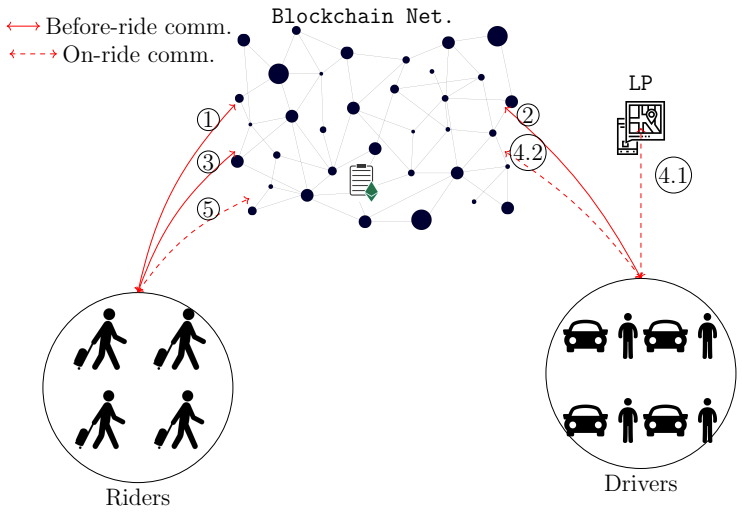
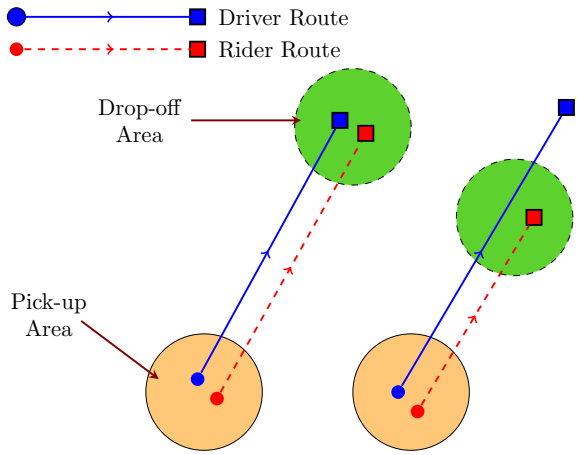


Figure 4.1: System architecture: (1) The rider publishes a ride request contract to the blockchain (2) Drivers sends their encrypted offers. (3) A rider selects the best matched offer and publish a time-locked contract. (4.1) and (4.2) Up on arrival, the driver sends a proof of arrival to pick-up and claim rider deposit (5) The rider publishes a payment contract that transfer the fare trip to the driver.

scheme to either act as a driver or rider. The ridesharing business logic is defined in a smart contract and executed by the blockchain network. The blockchain is also used for peer-to-peer payment to allow the exchange of currency between the different scheme users. Thus, we select Ethereum, which is the most popular open blockchain platform for smart contracts, to implement and evaluate our proposed protocol.

2. *Drivers and riders.* The rider is an entity that requests ride-sharing service by making requests. The driver is an entity that wants to share his trip by posting an offer to riders' requests.

3. *Location Prover (LP):* The role of LP is to ensure the authenticity of the reported location (pickup) by the drivers. In our scheme, upon arrival of a driver to a given pickup location, he/she should send its current location to the blockchain through the LP. An LP could be, for instance, roadside units that are already deployed to enable Vehicular Ad-hoc Networks



(a) Identical ride sharing (b) Inclusive ride sharing

Figure 4.2: Ride sharing cases under consideration in B-Ride.

(VANETs) [164], where each RSU can confirm if the location provided by a driver falls in its coverage or not.

Our scheme allows a small tolerance in pickup/drop off locations and times of the requests/offers [165]. Thus, spatial and temporal slacks are introduced to capture the maximum additional distance δ and time τ that a rider or driver can tolerate for traveling and waiting, respectively. In this chapter, we consider the following ride sharing cases:

1. *Identical ridesharing.* The pick-up and drop-off locations of both a driver and rider are matched, as shown in Fig. 4.2.a.

2. *Inclusive ridesharing.* The destination of the rider lies on the driver's route. In this case, the driver must stop for drop-offs before reaching its final destination, as shown in Fig. 4.2.b.

4.3.2 Threat and Adversary Assumptions

We assume that both internal and external adversaries could try to compromise the security of the ride sharing system. We follow the standard blockchain threat model presented in [1, 160],

where the blockchain is trusted for execution correctness and availability, but not for privacy. The smart contract code is visible and checkable by anyone once it is deployed and is guaranteed to work as specified. Likewise, any data submitted to the contract can be directly read by all parties of the system as well as any external curious users. In addition, the following threats are also considered:

- Global eavesdroppers can read all transactions recorded on the blockchain for the riders in order to learn their moving patterns, guess their locations at a specific time or even track them over the time.
- A rider may launch a location cheating attack by reporting a false planned trip to the blockchain. The rider fraudulently does not commit to his requests whereas the victim driver has to travel extra distance to pick up the rider. Likewise, a driver sends offers while deliberately not committing to them.
- Cheating in fare payment. If a driver gets paid at the beginning of the trip, he/she may misbehave and does not complete the trip. Also, if a driver gets the trip fare at the end of trip, the rider may not be willing to pay the fare [166].

4.3.3 Design and Functionality Requirements

Under the aforementioned system model and adversary assumptions, our objective is to develop a ride-sharing scheme with the following design goals:

1. *Resilience.* The proposed scheme should not rely on a central ride-sharing organizing entity to manage the ride-sharing service.
2. *Preserving riders' privacy.* The proposed scheme should preserve riders' privacy including their trip data, i.e., pick up/drop off locations. This can be satisfied if the following two conditions are achieved: (i) None of the drivers/miners, except the selected driver, learns the

exact location of the rider or the associated driver. (*ii*) A specific rider cannot be tracked over time.

3. *Ensuring trust between riders and drivers.* It is important to discourage any malicious behavior by both drivers and riders. This objective is challenging especially when relying on a public permissionless blockchain and also considering privacy concerns.

4. *Ensuring fair payment.* The proposed ride-sharing service should ensure fair payment in order to attract more users to the scheme. Hence, the payment should be done in a trust-less manner to protect honest drivers from dishonest riders and vice versa.

5. *Drivers reputation management.* The proposed scheme should keep track of drivers' behavior through a reputation system. Malicious drivers who may try to subvert the scheme, even irrationally, can be identified by their low reputation scores. Therefore, drivers having low reputation scores will be distrusted and no one will be interested to share ride with them.

4.4 Our Proposed Scheme: B-Ride

B-Ride consists of the following six phases: trip data generation, bidding and selection, time-locked deposit protocol, fair payment and reputation management.

4.4.1 Trip Data Generation

In this section, we discuss how drivers/riders generate their trip data while preserving their privacy. Generalization technique [39] also known as spatial cloaking, is used for this purpose. The main idea of the spatial cloaking is to blur users' exact locations into cloaked regions for location obfuscation. Let's denote by $\mathcal{A}$ the ride sharing area (e.g., a city) that is sub-divided into a set of *n cells* $C = \{C_1, C_2, ...C_n\}$. Cells could be defined based on geographic area constraints such as districts or neighborhoods in a city, uniform partitions in a map, etc. Therefore, instead of using the

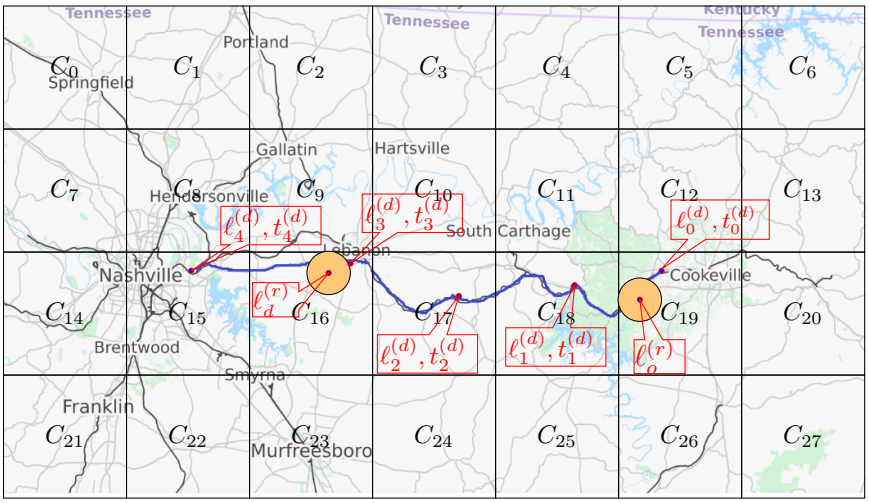


Figure 4.3: Illustration of dividing a ride sharing area of interest into cells. A driver d's route, in blue, with five points, and pickup and drop-off locations of a rider r (orange dots).

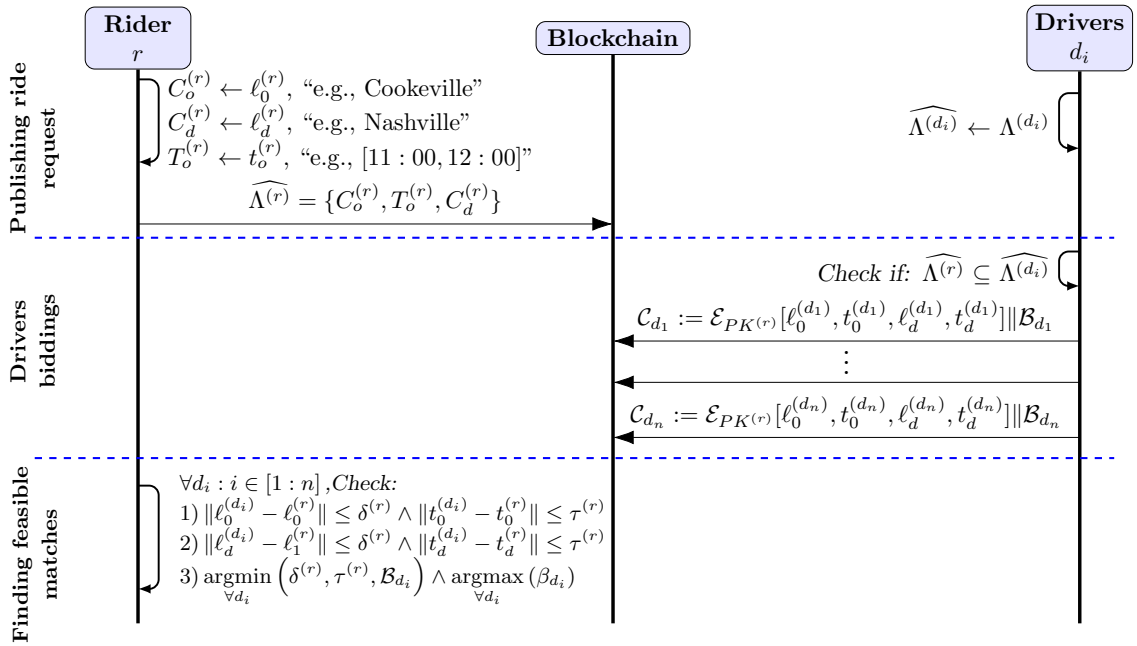


Figure 4.4: Illustration of the bidding and selection phase in B-Ride.

exact pick-up/drop-off locations, riders can only submit the respective cells containing the actual pick-up and drop-off locations. Fig. 4.3 illustrates an example of division the ride sharing area

into cells into 27 cells. Similarly, the exact pick-up/drop-off times can also be generalized (hidden) by using temporal cloaking where the actual time is generalized by setting *time interval* T. For instance, a driver d who is interested in sharing a ride with others should do the following steps before sending his offer.

1. The driver defines the planned trip $\Gamma^{(d)}$

$$\Gamma^{(d)} = \left\{ \left(\ell_0^{(d)}, t_0^{(d)}\right), \ldots, \left(\ell_k^{(d)}, t_k^{(d)}\right), \ldots, \left(\ell_n^{(d)}, t_n^{(d)}\right) \right\}$$

   that consists of the exact departure location $\ell_0^{(d)}$, departure time $t_0^{(d)}$, destination $\ell_n^{(d)}$ and estimated arrival time $t_n^{(d)}$, as well as a sequence of optionally intermediate locations and their corresponding arrival times $\left(\ell_k^{(d)}, t_k^{(d)}\right)$.

2. Then, he hides or cloaks his exact trip locations and times to zones (cells) and time intervals as follows.

$$\Lambda^{(d)} = \left\{ \left(C_0^{(d)}, T_0^{(d)}\right), \ldots, \left(C_k^{(d)}, T_k^{(d)}\right), \ldots, \left(C_n^{(d)}, T_n^{(d)}\right) \right\} \tag{4.2}$$

   where $\left(C_0^{(d)}, T_0^{(d)}\right)$ represents the cloaked location and time that corresponds to $\left(\ell_0^{(d)}, t_0^{(d)}\right)$.

3. Finally, he creates a set of all possible trips in his planned trip of the elements in $\Lambda^{(d)}$ as

$$\widehat{\Lambda^{(d)}} = \{\left(C_j^{(d)}, T_j^{(d)}, C_k^{(d)}\right)\}, 1 \leqslant j \leqslant n \text{ and } j+1 \leqslant k \leqslant n$$

   where the number of all possible trips depends on the number of chosen points n and it can be mathematically expressed as [167]

$$\begin{pmatrix} n \\ 2 \end{pmatrix} = \frac{n!}{2!(n-2)!}$$

For a rider r, we denote his ride request as $\Gamma^{(r)} = \left\{\left(\ell_0^{(r)}, t_0^{(r)}\right), \ell_d^{(r)}\right\}$, where $\left(\ell_0^{(r)}, t_0^{(r)}\right)$ represents the departure location and the desired set off time, and $\ell_d^{(r)}$ denotes the drop-off location. Similar to the driver, the rider also converts the request into a generalized form by mapping the pick-up and drop-off locations into the corresponding cells as well as departure time into cloaked time.

$$\widehat{\Lambda^{(r)}} = \{\left(C_o^{(r)}, T_o^{(r)}, C_d^{(r)}\right)\} \tag{4.3}$$

Note that all previous steps, including cloaking of the drivers' trip and the rider trip, are done off the blockchain using, for instance, the driver/rider smart phones.

4.4.2 Bidding and Selection

In this section, we describe the process of matching riders' requests with drivers' offers atop public blockchain. The smart contract that handles the B-Ride logic including bidding and selection is given in Algorithm 4.1. The different steps required for bidding and selection are illustrated by a schematic diagram in Fig. 4.4, and detailed in the following sections.

Publishing the Ride Request. First, as a fundamental concept to avoid de-anonymization in the blockchain, every rider r uses for each ride request a new blockchain address $\mathsf{ADD}^{(r)}$ that corresponds to a fresh public/private key pair $(PK^{(r)}, SK^{(r)})$. Then, a rider publishes a ride request that contains his/her cloaked pick-up/drop-off locations $(C_o^{(r)}, C_d^{(r)})$ and cloaked time $T_o^{(r)}$. Also, the request should include an expiry time to receive driver's offers. Optionally, the request can include a maximum number of offers to be received. Note that this request should be signed by the temporary private key of the rider and sent to the smart contract by calling the function *MakeRideRequest()* in Algorithm 4.1. Once the miners validate the corresponding signature of the rider, the request will be public to all drivers.

Algorithm 4.1: The Pseudocode for B-Ride Contract

```
Note that for the sake of explanation, we separated Algorithm 4.2 and
Algorithm 4.3.  In practice, B-Ride can be entirely implemented in a single smart
contract.
contract BRide
    mapping(address => int) Reputation score_1 // Mapping for drivers' reputation
       score of arriving to pick-up
    mapping(address => int) Reputation score_2 // Mapping for drivers' reputation
       score of completed trips
    contract BiddingSelectionContract
        function MakeRideRequest(C_0^(r), C_d^(r), T_0^(r))
            // Receive ride request.
        function MakeRideOffer(C^(d_i), B_{d_i})
            // Receive ride drivers biddings.
    contract TimeLockedDeposit
        Create a sub-contract of Algorithm 4.2
    contract PaymentContract
        Create a sub-contract of Algorithm 4.3
```

Submitting Drivers' Offers. For a driver that wants to share a ride, he first needs to receive a public key certificate from the corresponding registration authority (RA), such as the government. A driver d having a unique identity (e.g., license plate number), creates a public-secret key pair $(PK^{(d)}, SK^{(d)})$ and registers at the RA to obtain a certificate binding $PK^{(d)}$ to d.

Drivers can either periodically query the blockchain to obtain new riders' requests or use some out-of-band signaling protocols to get notified each time a new request is published [168]. In order to make an offer, a driver d first checks if the spatio-temporal date of a received request made by a rider r matches one of his planned trips, (i.e., $\widehat{\Lambda^{(r)}} \in \widehat{\Lambda^{(d_i)}}$). If the driver finds a ride request in his route, as shown in Fig. 4.4, the driver creates an offer that includes all the necessary information for the rider such as, the exact pick-up location and time $\left(\ell_0^{(d_i)}, t_0^{(d_i)}\right)$, the exact drop-off location and time $\left(\ell_d^{(d_i)}, t_d^{(d_i)}\right)$ as well as the offer bid price $\mathcal{B}_{d_i}$ (e.g., price per mileage). Then, the driver uses the rider's public key to encrypt all above information to obtain $\mathcal{C}_{d_i}$

$$\mathcal{C}_{d_i} = \mathcal{E}_{PK^{(r)}}\left(\ell_0^{(d_i)}, t_0^{(d_i)}, \ell_d^{(d_i)}, t_d^{(d_i)}\right),$$

where $\mathcal{E}$ is an asymmetric encryption, e.g., RSA. This encryption is necessary to hide the trip information from the blockchain and eavesdroppers to preserve the privacy of both the driver and rider. The tuple $\mathcal{C}_{d_i} \| \mathcal{B}_{d_i}$ is sent by the driver to the smart contract by calling the function *MakeRideOffer()*. Note that the bidding price is not encrypted to ensure price competition which guarantees low prices for the riders. Also since the rider request includes an expiry time to receive offers, any offer made after that time will be automatically rejected by the blockchain network.

Finding Feasible Matches. After receiving, for instance, n offers $\{\mathcal{C}_{d_1}, \cdots, \mathcal{C}_{d_n}\}$ for the same request made by a rider r, the rider retrieves the encrypted offers from the smart contract and decrypts each of them off the chain using his secret key. In order to select an offer that better matches the rider preferences, the offers are evaluated as follows.

1. The rider checks if the driver's pick up and drop off data are matched *spatially* by checking if:

$$\left(\ell_o^{(d_i)} - \ell_o^{(r)} \leq \delta^{(r)}\right) \wedge \left(\ell_d^{(d_i)} - \ell_d^{(r)} \leq \delta^{(r)}\right), \tag{4.4}$$

   where $\delta^{(r)}$ is the maximum tolerated distance between the rider's exact pick up (or drop off) location and driver's pick up (or drop off) location. Also, the rider checks if the driver's pick up and drop off data are matched *temporarily* by checking if:

$$\left(t_o^{(d_i)} - t_o^{(r)} \leq \tau^{(r)}\right) \wedge \left(t_d^{(d_i)} - t_d^{(r)} \leq \tau^{(r)}\right), \tag{4.5}$$

   where $\tau^{(r)}$ is the maximum tolerated time difference between the rider's pick up (or drop off) time and the driver's pick up (or drop off) time.

2. Besides, in B-Ride, each driver d has a reputation score β_{d_i} that is stored on the blockchain. The detail about how each driver's reputation score is updated will be discussed in details in the next section. Using $\delta^{(r)}$, $\tau^{(r)}$, $\mathcal{B}_{d_i}$ and the drivers' reputation score, the rider is able to

select the best offer that matches his preference as follows:

$$\underset{\forall d_i}{\operatorname{argmin}}(\delta^{(r)}, \tau^{(r)}, \mathcal{B}_{d_i}) \wedge \underset{\forall d}{\operatorname{argmax}}(\beta_{d_i})$$

Note that preference may vary from one rider to other. For instance, some may prefer an offer with a low price even if it has a high distance slack $\delta^{(r)}$ or waiting time $\delta^{(r)}$. Different from existing centralized approaches, finding feasible ride matches is handled over the blockchain and is therefore, fully transparent.

4.4.3 Time-locked Deposit Protocol

In this section, we introduce the time-locked deposit protocol that penalizes malicious drivers or riders that do not commit to their ride offers or requests. Usually, a traditional solution to this problem is to allow both parties (i.e., drivers and riders) to pay a subscription fee to a trusted agency that can be contacted whenever a breach occurs. However, if the third party gets attacked, the whole system becomes unprotected against malicious users.

Inspired by [169], we propose a time-locked deposit protocol for ride-sharing service leveraging smart contracts and blockchain. The idea is that the smart contract receives a deposit from both rider and driver and conditionally transfers the total deposit amount to the driver if he/she arrives to the rider pick-up location at the predefined agreed time. However, if the driver defaults, the deposit will be transferred to the rider after a prespecified time as a fine on the driver. Note that the conditions are defined in the smart contract and executed over the blockchain, in a completely transparent and secure manner. Nevertheless, the definition of these conditions should consider the privacy of the drivers and riders. In other words, how a driver can prove that he/she has arrived to the rider's pick-up location without revealing such sensitive information on the public blockchain. For this purpose and in order to preserve users' privacy, we leverage the use of the zero-knowledge set membership proof (ZKSM) protocol presented in Section 4.2.2. In nutshell, after selecting an offer

of a driver, the rider publishes a new *time-locked deposit* smart contract (see Algorithm. 4.2), and initializes it by sending the trip deposit. The corresponding driver of the selected offer should act as a *prover* who needs to demonstrate to the blockchain (*verifier*) in a *zero-knowledge* manner that he/she has arrived to one of the pick-up locations already published by the rider without revealing the exact pick-up location. A schematic diagram of the time-locked deposit protocol is depicted in Fig. 4.5.

In the following, we describe the different steps of the time-locked deposit protocol.

Initialization. The rider initializes the time-locked deposit protocol as follow:

1. Defines a set ϕ of k locations where $\phi = \{\ell_1, \cdots, \ell_k\}$. The set ϕ should include the actual rider pick-up location $\ell_o^{(r)}$, as well as other obfuscated locations.

2. Picks a random number $x \in_R \mathbb{Z}_p$ and computes the corresponding public $y \in g^x$.

3. Computes for every element $i \in \phi$ the corresponding signature (A_i) using g and x where, $A_i = g^{\frac{1}{(x+i)}}$.

4. Publishes a new smart contract, given in Algorithm 4.2, and calls the function *TimeLockedDeposit(ϕ, A_i, deposit)*.

Driver Deposit and Claim. In this step, the driver needs to confirm his/her commitment to the trip offer by sending a deposit to the smart contract. The driver can claim back the deposit, once he/she arrives to the pick-up location, by providing a proof of arrival to that location at the pick-up time.

For this purpose, before sending the deposit, the driver first authenticates each element in ϕ using the corresponding rider's public key[3]. The verification is done by checking the following

[3]The purpose of this verification is to prevent a malicious rider from falsely setting-up the ZKSM protocol so he/she cannot manipulate the rewards of the driver.

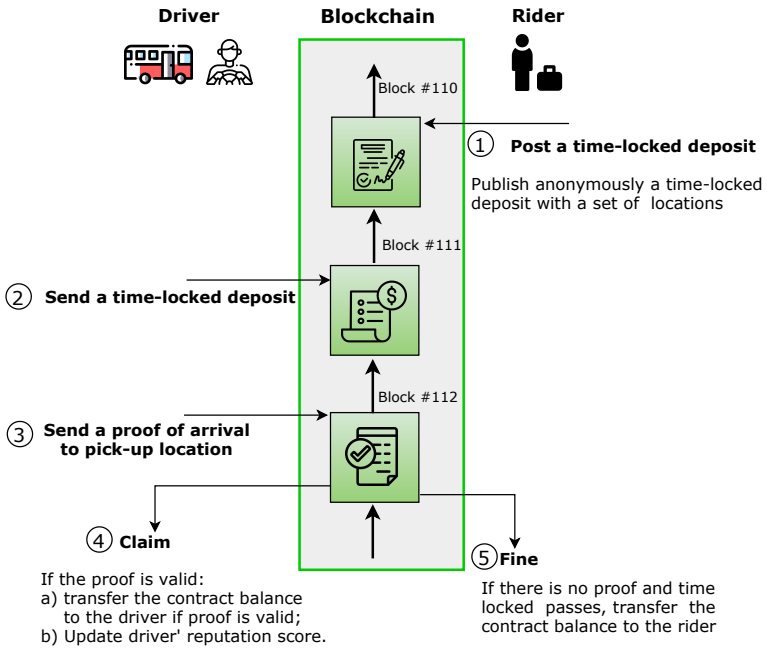


Figure 4.5: The schematic diagram of the time-locked deposit protocol.

equality [170]

$$e\left(\mathcal{A}_i, y \cdot g^{\ell_i}\right) \stackrel{?}{=} e(g, g)$$

To send the deposit, the driver calls the function *DriverDeposit($\mathcal{D}$)*, where $\mathcal{D}$ is a deposit to the contract within a limited window of time (See line 15 in Algorithm 4.2).

When the driver reaches the pick-up location, he/she needs to prove in a zero-knowledge manner that he/she arrived to the pick-up location as follows:

1. The driver picks $v \in_R \mathbb{Z}_p$ and computes $V = \left(\mathcal{A}_{\ell_o^{(r)}}\right)^v$, where $A_{\ell_o^{(r)}}$ is the rider' signature on $\ell_o^{(r)}$. Then, he/she computes a commitment on $\ell_o^{(r)}$ using Pedersen commitment [171] as follws: $C = g^{\ell_o^{(r)}} h^{\iota}$, where ι is a random number and h is a random group element such that it is hard to find the discrete logarithm of g base h. The tuple $C \| \ell_o^{(r)} \| \sigma_\iota(\ell_o^{(r)})$ is then sent to the location prover (LP) where $\sigma_\iota(\ell_o^{(r)})$ is the driver' signature on $\ell_o^{(r)}$.

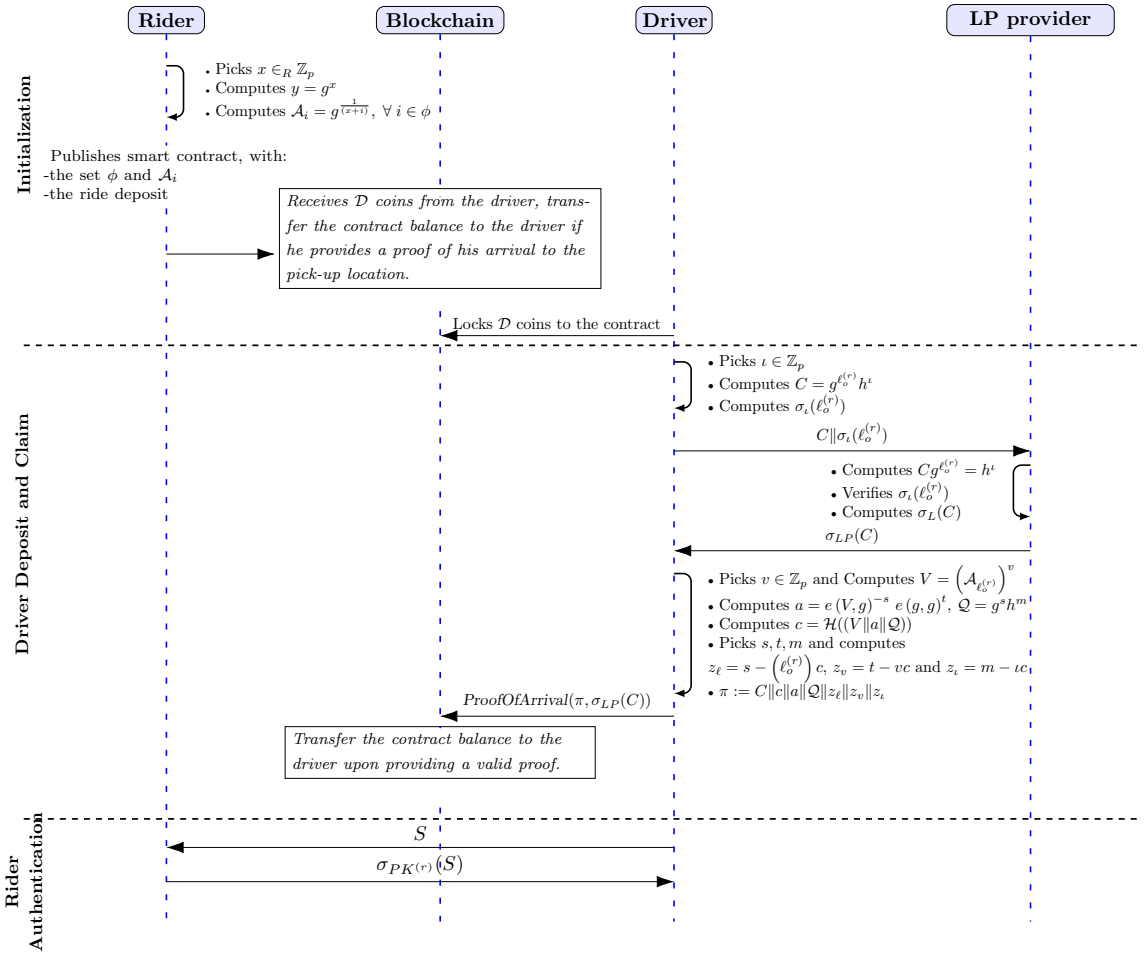


Figure 4.6: Time-locked deposit protocol in B-Ride.

2. The LP verifies if the received location from the driver is authentic. The LP Computes

$$C' = Cg^{-\ell_o^{(r)}} = g^{\ell_o^{(r)}} h^\iota g^{-\ell_o^{(r)}} = h^\iota$$

where h^ι is the public key corresponding to ι. Then, the LP verifies if the signature $\sigma_\iota(\ell_o^{(r)})$ is valid using h^ι. The tuple $C \| \sigma_{LP}(C)$ is then sent back to the driver, where $\sigma_{LP}(C)$ is the LP's signature on C.

3. The driver picks random numbers $s, t, m \in_R \mathbb{Z}_p$ and computes $a = e(V,g)^{-s}\, e(g,g)^t$ and

Algorithm 4.2: The Pseudocode for Time-Locked Deposit Contract in B-Ride

```
contract TimeLockedDeposit
    uint public Balance // Balance to withhold driver and rider deposits
    address payable rider // Rider address
    address payable driver // driver address
    uint public RiderDeposit // RiderDeposit
    uint public DriverDeposit // DriverDeposit
    address [] Set // set of obfuscated locations
    address [] A_i // Signatures of elements in the set
    // Constructor
    function TimeLockedDeposit(_driver, _Set, _A_i, _RiderDeposit)
        driver ← _driver; // The address of selected driver
        Balance ← _RiderDeposit ;// This deposit acts as acceptance to the driver offer.
        Set ← _Set;
        A_i ← _A_i;
    function DriverDeposit(uint256 _DriverDeposit)
        // Receive Driver Deposit and add to the contract
        if block.timestamp ≥ expiration  return;
        if msg.sender ≠ DriverAddress  return;
        if msg.value ≠ DriverDeposit  return;
        if now ≥ T^{Accept}_{deadline}  return;
        // T^{Accept}_{deadline} is a window for the driver to send his/her deposit e.g., 2 min
        Balance ←_DriverDeposit;
    function ProofOfArrival((π, σ_{LP}(C)])
        // π = {D‖c‖a‖Q‖z_σ‖z_v‖z_r}
        // If the driver provides a valid proof of the agreed pickup location, he/she will be rewarded
           by the rider deposit and he/she get his/her deposit back.
        if msg.sender ≠ DriverAddress  return;
        if now ≠ T_o^{(r)}  return;
        // Check the time of receiving the proof lies in the generalized rider pick-up time
        if (ZKSM.Verifier(π))
            // ZKSM.Verifier is a library embedded in the runtime environment of smart contract such as
               EVM
            BRide.Reputation score_1[msg.sender]← Reputation score_1[msg.sender]+1; // Increase the
               driver reputation score of arriving to pick-up location
            // call public event Transfer to finalize the payment
            transfer(balance, driver);
        end
    function FineDriver()
        // Issue a rider deposit back and the driver deposit and if the timeout has expired.
        if block.timestamp < expiration  return;
        if msg.sender ≠ rider  return;
        transfer(balance, rider); // Transfer the contract balance back to the rider account.
```

$\mathcal{Q} = g^s h^m$. Note that the adopted ZKSM in [162] requires an interactive session between the prover and the verifier with multiple rounds of communication. However, in the context of blockchain, the miner can not properly agree on a common value of proof-related parameters, such as the challenges, since they need to be chosen randomly. One solution is to use the FiatShamir heuristic [172], which is a generic technique that allows to convert interactive zero-knowledge schemes to non-interactive protocols. Using the Fiat-Shamir heuristic, a challenge c can be calculated from public parameters as follows

$$c = \mathcal{H}(V \| a \| \mathcal{Q})$$

Then, he/she computes $z_\ell = s - \left(\ell_o^{(r)}\right) c$, $z_v = t - vc$ and $z_\iota = m - \iota c$.

Then, the proof is denoted as $\pi = C\|c\|a\|\mathcal{Q}\|z_\ell\|z_v\|z_\iota$. Finally, he/she sends $\pi\|\sigma_{LP}(C)$ as a transaction to the blockchain.

4. Once the contract receives the proof π, it verifies whether (i) the proof is from the selected driver, and (ii) the time of receiving the proof lies on the generalized rider pick-up time (see lines 19-20 in Algorithm 4.2). The proof that C is a commitment to an element in ϕ can be validated by the following statement [162].

$$\mathbf{PK}\left\{(\ell_o^{(r)}, \iota, v) : C = g^{\ell_o^{(r)}} h^\iota \wedge V = g^{\frac{v}{x+\ell_o^{(r)}}}\right\} \tag{4.6}$$

Proving the statement in (4.6) is done in the blockchain by checking the following conditions:

$$\mathcal{Q} \stackrel{?}{=} C^c h^{z_\iota} g^{z_\ell} \tag{4.7}$$

$$a \stackrel{?}{=} e\left(V, y\right)^c \cdot e\left(V, g\right)^{-z_\ell} \cdot e\left(g, g\right)^{z_v} \tag{4.8}$$

Once the function *ProofOfArrival* validates the two conditions in 4.7 and 4.8, the driver claims the contract balance (see lines 20-26 in Algorithm 4.2). Due to the *Soundness* and *Completeness* property of ZKSM [162], the contract accepts the proof if it is correctly constructed by the driver. If the driver breaks his/her commitment to arrive in the pick-up time, which is pre-defined in the time-locked deposit contract, he/she will be fined with a monetary penalty that goes to the rider' account (see lines 27-30 in Algorithm 4.2).

Driver/Rider Authentication. In order to prevent impersonation attacks, in which a malicious rider tries to take a ride that was reserved by another rider, driver and rider must authenticate each other. Specifically, the rider should prove to the driver in a zero knowledge manner that he/she

indeed knows the value of private key corresponding to the public key ($PK^{(r)}$) that made the reservation. The driver (the verifier) selects a uniformly random integer $\mathcal{S} \in \mathbb{Z}_p$ as a *challenge* and sends it to the rider (the prover). The rider uses his/her private key to generate a signature on the challenge ($\sigma_{SK^{(r)}}(\mathcal{S})$) and sends ($\mathcal{S} \| \sigma_{SK^{(r)}}(\mathcal{S})$) to the driver. Finally, the driver verifies if $VerifySig(PK^{(r)}, \sigma_{SK^{(r)}}(\mathcal{S}), \mathcal{S}) = 1$.

4.4.4 Fair Payment

In this section we address the following challenge: *how to estimate the fare of each trip without trusting riders and drivers.* In current RSSs, to estimate the trip fare, a driver needs to report the distance and duration of each ride using his/her smartphone to the service provider. However, smartphones are general-purpose devices and are easy to tamper with. Thus, a malicious driver can report longer ride distances to get higher fare. This problem is usually referred to as *overcharging* [109], [166].

In B-Ride, we tackle the above challenge by adopting the *pay-as-you-drive* method. After starting the trip, the rider makes a call to the *fare payment* smart contract (see Algorithm 4.3) and initializes it with a sufficient amount of coins as a deposit to be used later for the payment of the fare. Note that the down payment used in the time locked deposit protocol is also used as a part of the payment of trip's fare. After that, for every period of time, the driver sends the elapsed distance to the rider. The rider checks whether the received distance matches the actual elapsed distance or not. Using multi-signature scheme, the elapsed distance is signed by both the driver and the rider, and then sent to the smart contract as a proof of the actual elapsed distance approved by the driver and rider. Thus, a payment that corresponds to this travelled distance, goes to the driver account. This ensures that the driver is paid based on the distance that has been actually travelled. In case a malicious rider stops sending the proofs of the elapsed distance, the driver can decide to stop the ride without affecting his/her payment on the previous travelled distances. Similarly, if the driver

Algorithm 4.3: The Pseudocode for the fair payment contract in B-Ride.

```
contract RidePayment
    address payable rider // Rider address
    address payable driver // driver address
    uint public TripDist // driver address
    // Constructor
    function RidePayment(_driver, _TripDist, _t_d^(R))
        _driver ← _driver // A greed distance of payment
        TripDist ← _TripDist // A greed distance of payment
    function ProofOfDistance(ElapsedDist)
        // If this is called with the rider, the driver gets paid.
        if msg.sender ≠ RiderAddress  return;
        while TripDist ≤ ElapsedDist  do
            transfer(balance × ElapsedDist), pk_d) // Decrease balance according to the travelled
               distance by the rider and driver
            TripDist ← _TripDist-Elapsed_Distance ;
            if (TripDist==0)
                B-Ride.Reputation score_2[DriverAddress]← Reputation score_2[DriverAddress]+1;
                // Update the reputation driver score of completed trips
            end
        end
    function withdrawFunds()
        // Issue a refund back to the rider if the timeout has expired.
        if block.timestamp < expiration  return;
        if msg.sender ≠ owner  return;
        transfer(balance, owner) // Transfer the contract balance back to the rider account.
```

decides to stop the trip, the rider can stop sending the elapsed distance proof and, therefore, the driver will not be paid. Finally, if the driver does not complete the trip in the pre-agreed time of the driver's offer, the remaining payment can goes back to the rider's account (see function Refund in algorithm 4.3). This encourages the driver to commit to his/her offer and complete the journey on time without delay.

4.4.5 Reputation Management

In B-Ride, before starting the trip, the driver needs to prove his/her arrival to the pickup location, and the rider is required to prove his/her good willing by making a down-payment. However, in case the ride was not performed, we cannot determine which side is circumventing (uncommitted to the trip), whether the rider or driver. In addition, because the down-payment made by a rider automatically goes to the driver once he/she proves his/her arrival to the pick-up location, a dishonest driver can collect the down-payments without committing to his/her offer. Also, since the riders are anonymous, it is not possible to make a claim about such situation. To mitigate this

problem, we propose a reputation score for drivers that records any similar incidents as a potential driver misbehavior. If for the same driver this case happens many times, the reputation score of the driver reduces and this will discourage riders from accepting any offers coming from that driver.

More specifically, each driver is assigned a reputation score that is considered by riders during the selection process. A high reputation score reflects the good driver behaviour. Moreover, unlike traditional ride sharing schemes where the reputation is managed and controlled by a third party, our reputation is completely decentralized and implemented on blockchain. Each driver is tagged with a reputation score β_d. The score has two values, the first value β_D^{AP} increments every time the driver proves his/her arrival to the agreed pick up location, and the second one β_d^{AD} increases for every received proof of completing the trip. The driver's reputation β_d^{AP} increases and is recorded in blockchain if the function `ProofofArrival` validates his/her proof of arrival to the rider pick up location (see line 24 in Algorithm 4.2). Meanwhile, the driver's reputation score β_d^{AD} increases only if the trip is completed. (see line 14 in Algortihm 4.3). Therefore, when a dishonest driver d tries to collect riders deposits without completing the corresponding trips, the β_d^{AP} increases while β_d^{AD} keeps decreasing and becomes smaller than β_d^{AD}. Thus, the final reputation score β_d of driver d can be estimated as follow:

$$\beta_d = \frac{\beta_d^{AP}}{\beta_d^{AD}} \tag{4.9}$$

The value of the reputation score allows to distinguish between three different cases: (i) $\beta_d = 1$ means that $\beta_d^{AP} = \beta_d^{AD}$. In this case, the driver is considered honest as he/she has committed to all his/her past trips; (ii) $\beta_d > \mathcal{T}$, where $\mathcal{T}$ is a threshold value pre-defined by the riders. In this case, the driver is considered dishonest and the scheme refunds all riders from whom he/she took the down-payment deposit, and (iii) $\mathcal{T} < \beta_d < 1$. In this case, the driver is still considered honest but riders may prefer to select drivers with better score for their future trips.

4.5 Performance Evaluations

In this section, we evaluate the performance of B-Ride.

4.5.1 Implementation and Performance Metrics

Fig. 4.7 shows an overview of the proof-of-concept implementation of B-Ride. Usually, anything executed in the blockchain environment is refereed as being *on-chain*, while anything that runs outside the blockchain is *off-chain*. All the operations required to setup ZKSM protocol, as well as the cloaking of trip data, encryption of the offers and generalization of the trip request data, are made off-chain. Whereas the execution of the B-Ride smart contract and the related transactions are performed on-chain.

We have implemented a smart contract for Algorithm 4.1 in Solidity [4]. B-Ride [5] is deployed into the public Kovan Etehreum test network [173]. In B-Ride, verifying the ZKSM needs some intensive calculation in order to run mathematical and cryptographic functions required by the time-locked deposit protocol. However, implementing these functions in a high-level language, such as Solidity, would be costly in terms of gas. To mitigate this issue, we have deployed a *precompiled contract* that implement these functions. By using a precompiled contracts, less gas is required, as the code is not run on the EVM, but rather on the machine hosting the Ethereum client [174]. Typically, a precompiled contract has a fixed address and gas price, and can be invoked using the call operation. Table 4.3 gives the precompiled contracts used in this chapter.

Ethereum introduced the concept of *gas* to quantify the cost associated with each transaction. The cost is payable using the native Ethereum currency, named *Ether*. Each operation in a smart contract has a fixed cost. For instance, the addition of two variables requires 3 gas, multiplication

[4] https://github.com/ethereum/solidity

[5] We note that B-Ride contract in Algorithm 4.1 is coded as a single "factory contract" in which upon receiving a message with the variable arguments from both entities i.e., riders and drivers, will create a new instance of the child contracts Algorithm 4.2 and Algorithm 4.3

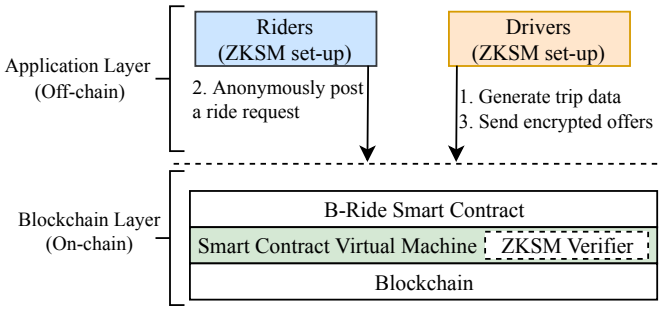


Figure 4.7: Implementation overview of B-Ride.

costs 5 gas has computing a SHA3 hash needs 30 gas plus 6 gas for every 256 bits of input [2]. Therefore, to evaluated the cost of the on-chain operations, we are interested in the following metrics.

- *Transaction cost* is measured by the overall gas cost of sending data to the blockchain. Typically, there are four items which make up the full transaction cost as follows: (i) the base cost of a transaction, (ii) the cost of a contract deployment, (iii) the cost of the code in a transaction [2].

- *Execution cost* indicates the portion of gas that is actually spent on running the code included in a transaction by the Ethereum Virtual Machine (EVM) [175].

- *Storage cost* is the cost of storing the data in blockchain.

Previous metrics can be converted into direct monetary cost on both drivers and riders, which can help to assess the practicality of running our scheme on public blockchain. In the following, we evaluate the on-chain and off-chain costs in B-Ride.

4.5.2 On-chain Cost

Fig. 4.8 gives the gas consumption per trip costs on the rider side. About 400K gas is required for publishing the bidding contract. Then, 80K gas is required to retrieve the drivers' offers. 320K

Table 4.3: The precompiled contracts used in B-Ride

Operation	Address	Operation	Description
ECADD	0x06	500	Elliptic curve addition.
PAIRING	0x08	100.000 + 80.000 $\times k$	Optimal ate pairing check.
ECMUL	0x07	40.000	Elliptic curve scalar multiplication.

Note that in the PAIRING, k denotes the number of points or, equivalently, the length of the input divided by 192.

gas is needed to deploy the time-locked deposit contract. 340K gas is required for deploying the payment contract, and 42K gas is needed for calling the `ProofOfDistance` function to perform the fare payment.

Fig. 4.9 gives the gas consumption required by a driver to complete a trip. 89K gas is required to send the encrypted offers to the bidding contract. Once the driver is selected by a rider, 25K gas will be used to send the driver's deposit to the time-locked contract. Finally, the validation of the proof of the underlying ZKSM requires 360K gas.

Fig. 4.10 gives the estimated total cost of driver versus the number of riders, given different gas prices and Ether price of $217 [176]. Having 40 trips, the driver costs about $2. By increasing the gas price, the driver costs increase to reach $7.6 for completing 40 trips. The results clearly show that the cost is low and very affordable for the end users.

Fig. 4.11 gives the storage overhead in *Bytes* on the blockchain. It can be seen that the associated storage cost is low and practically acceptable. For instance, with 7 submitted offers, the storage on the blockchain is about only 12 KBytes.

4.5.3 Off-chain Overhead

We evaluate the execution time of the off-chain operations on both the driver and rider side. Raspberry Pi 3 device with 1.2 GHz Processor and 1 GB RAM is used to emulate the driver/rider devices. We used BN128 pairing-friendly elliptic curves, that is available on Github [177], to estimate

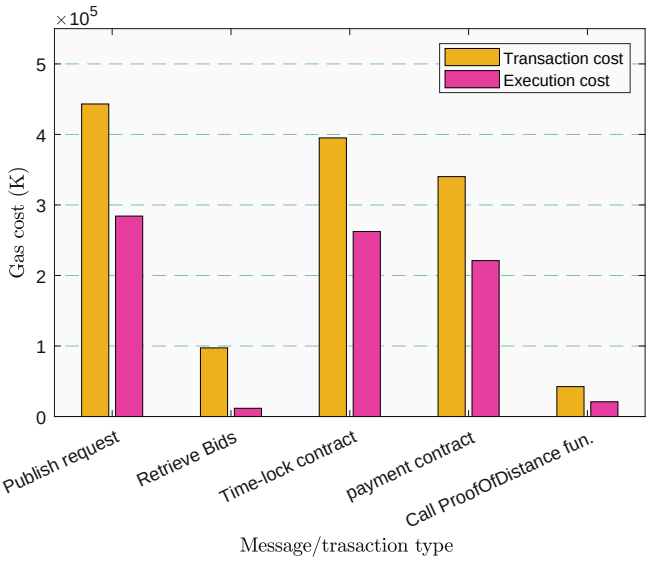


Figure 4.8: The estimated gas cost of calling contract functions on Kovan test net on the driver.

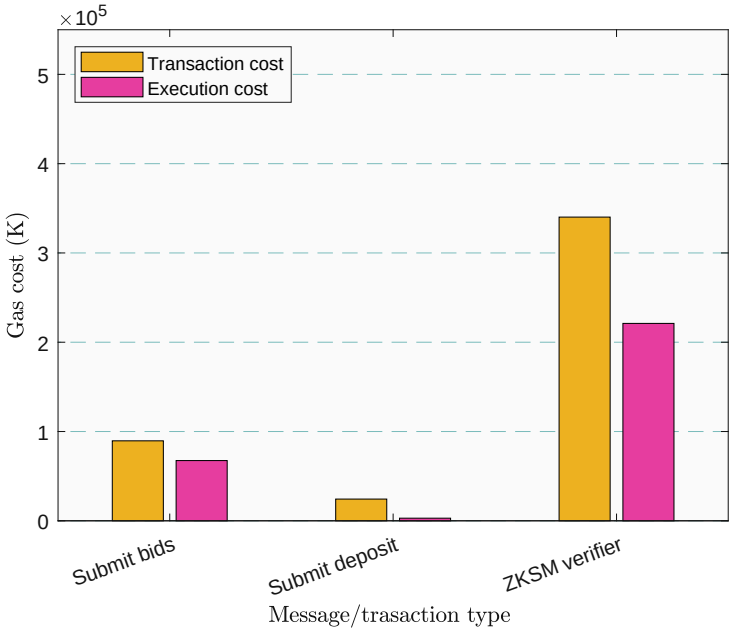


Figure 4.9: The estimated gas cost of calling contract functions on Kovan test net on the rider.

the computation overhead of the ZKSM protocol. The obtained results are summarized in Table 4.4. It can be seen that the execution time of the off-chain operations is very short as it takes less than a second.

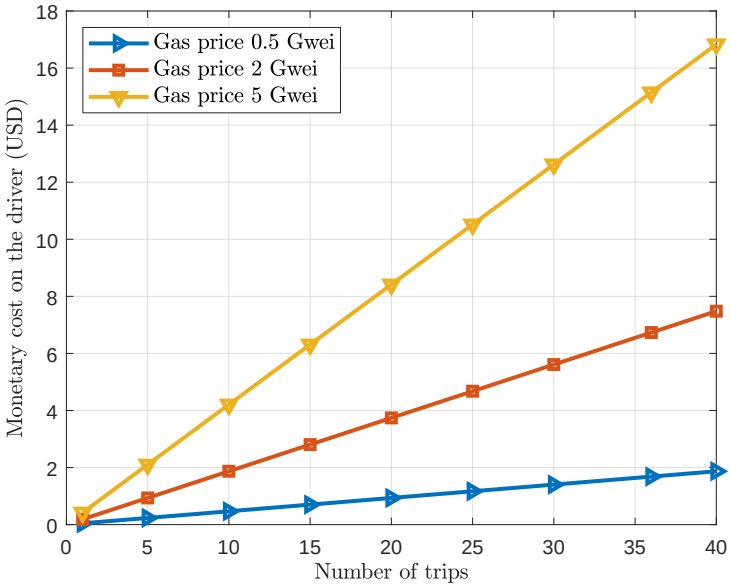


Figure 4.10: The momentary cost of the drivers versus the number of trips.

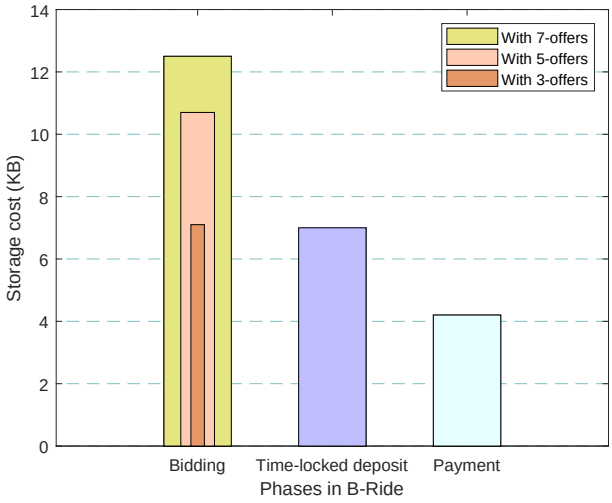


Figure 4.11: On-chain storage cost in B-Ride.

4.6 Security and Privacy Analysis

In this section, we discuss security/privacy analysis of our scheme. Specifically, our schemes can achieve the following security/privacy objectives.

Security and transparency. Our scheme is secure against single point of failure attack due

Table 4.4: Off-chain overhead

Operations	Involved entity	Time (s)
ZKSM setup	Rider	0.5
ZKSM Proof	Driver	0.9

to using the blockchain (a distributed network) to run them instead of using a central server. Moreover, due to the immutability nature of the blockchain, the data stored on the blockchain cannot be tampered with by malicious adversaries. Also, our schemes ensure *transparency* of ride sharing activities. This is because unlike centralized systems in which it is not known how the server runs the schemes so it can favor drivers over others to share their trips first, in our schemes, all transactions are posted on the blockchain and the drivers/riders can ensure that their offers/requests are received and handled properly.

Preserving rider's privacy. To ensure privacy of riders, the rider uses a random blockchain address that acts as a *pseudonym* and the generation of that address is unlinakable to the rider real identity. Also, the rider's trip data including pickup/drop-off locations and the pickup time are protected using the cloaking technique. Also, the drivers' offers are encrypted by the rider's public key and thus they are *confidential* to anyone including the blockchain miners or other attackers, and only the rider can decrypt and obtain the submitted offers and no information is leaked to the public. Finally, due to the zero-knowledge property of the underlying ZKSM, the blockchain can verify the proof of arrival to the pickup location while not knowing that location.

Security against a malicious driver. The ways that malicious drivers can cheat are: (*i*) submitting multiple offers to deliberately let riders have fake reserved trips to make the scheme unreliable; (*ii*) providing a valid proof to pick-up location to claim the rider's deposit while not starting the trip with him; and (*iii*) overcharging the rider in the fare payment. The first threat is thwarted since the driver has to add a deposit to the time-locked deposit contract within a specific time. If the driver does not provide a proof of arrival to the pick-up location, he/she would lose his

deposit. The second threat is discouraged since our scheme builds a reputation score for each driver to track and monitor the behaviour of the drivers. By having a reputation score for each driver, drivers who behave honestly have good score and will be selected by riders, while drivers who do not commit to the trips have a low reputation score and will not be selected in the future trips. The third threat is prevented by allowing the rider to check the distance provided by the driver before the fare is paid.

Security against malicious riders. A malicious rider misbehaves by three ways: (i) submitting multiple requests to intentionally make fake reservations; (ii) cheating in the ZKSM set-up phase by providing a set ϕ with fake signatures of the set elements in the ZKSM in order to deter the driver from claiming his/her trip deposit; (iii) cheating in *payment* phase by providing a fake elapsed distance that is shorter than the distance that is actually travelled. The first case is mitigated since a driver accepts to share a trip with a rider only if the later adds a ride-deposit to the time-locked deposit smart contract. The second threat is prevented because the time-locked smart contract is public, and the driver can validate the ZKSM set-up using the digital signatures. The last threat is prevented by the use of the public blockchain that enables the driver to check whether the rider provides a valid elapsed distance to the payment contract.

CHAPTER 5

BLOCKCHAIN-BASED CHARGING COORDINATION SCHEME FOR SMART GRID ENERGY STORAGE UNITS

In this chapter, we present our blockchain-based charging coordination scheme for smart grid energy storage units.

5.1 Overview and Main Contributions

In this chapter, we propose charging coordination scheme on top of public blockchain. A smart contract defining the rules to coordinate charging between different ESUs is proposed. The idea is that each ESU sends a charging request containing its power demand, state-of-charge (SoC), and time-to-complete-charging (TCC) to the smart contract. To preserve privacy, each ESU has a number of *certified pseudonyms* and each pseudonym is used only for one charging request. The charging request is anonymously signed and this anonymous signature can prevent external attackers from sending valid charging requests. Based on the submitted data and the maximum available charging energy capacity, the smart contract computes a priority index for each ESU, and ESUs with the highest priority can be charged using a Knapsack algorithm [178]. To be the best of our knowledge, *this work is the first to use the blockchain technology to enable decentralized charging coordination scheme* with the following features:

1. *Reliability.* The scheme is reliable because of running the charging coordination scheme does not rely on a trusted server, called charging controller (CC), and all the computations are carried out in a decentralized manner through the blockchain.

2. *Privacy-preservation.* Since the ESUs use anonymous credentials, no one can link the SoC and TCC to an ESU that sent them which can preserve the privacy of ESUs' owners.

3. *Transparency and verifiability.* Due to using blockchain, the charging requests and schedules are posted on the ledger, so ESUs can run Knapsack algorithm to ensure that the charging schedules are calculated correctly.

4. *Data integrity.* Once ESUs send their charging requests with TCC and SoC to the blockchain, they can later ensure the integrity of the requests' data since they have access to the blockchain, which cannot be done in centralized approaches.

The rest of this chapter is organized as follows. In Section 5.2, we discuss some preliminaries. Then, we introduce blockchain-based charging coordination scheme in section 5.3. The evaluations of the proposed scheme are provided in Section 5.4.

5.2 Preliminaries

Blind signatures and Partial Blind Signatures (PBS) have been extensively used in the anonymization of electronic coins, and were introduced by [179]. Indeed, Blind Signature [14] allows the sender of a message to obtain signature on this message from another party while concealing the content of the message. The signature requester generates a secret pair of blinding/unblinding operations (b, b^{-1}), and applies the blinding operation b to a plaintext message m. Then, he/she sends the blinded message $b(m)$ to the signer which signs it with operation s and produces a signature $s(b(m))$, and returns the blinded signature to the requester. Note that during this phase the requester uses its real identity. The requester then uses the unblinding operation b^{-1} to the blinded signature to obtain $b^{-1}(s(b(m))) = s(m)$, which is a signature on the plaintext message. Partial Blind Signature (PBS) is a special case of Blind Signature where the signer can include a common message m_0 that is known to both singer and sender, such as a time or date. The requester submits

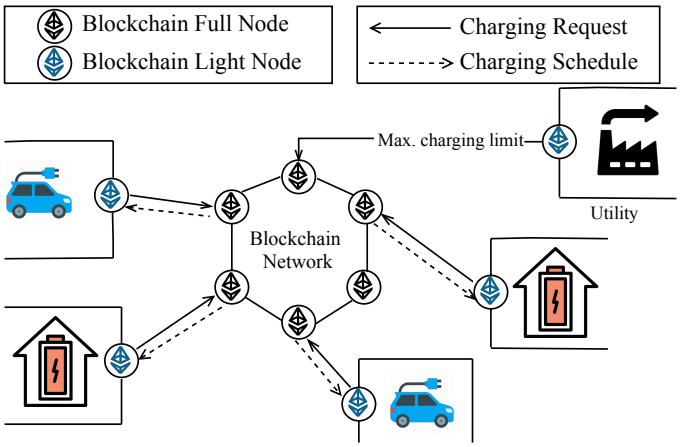


Figure 5.1: Illustration for the system model under consideration.

blinded message $b(m)$, the signer generates a blinded signature $s(b(m), m_0)$, and returns it back to the requester. The requester applies the unblinding operation to get $b^{-1}(s(b(m), m_0)) = s(m, m_0)$ which is the signature of the signer on $m \| m_0$. Note that the signer only knows the blinded message $b(m)$ and can not know the plaintext message m. The requester can unblind a signature $s(b(m), m_0)$ to obtain $s(m, m_0)$, and can verify that $s(m, m_0)$ is a valid signature on (m, m_0), but it cannot forge a signature. Also, the signer can not link $s(m)$ (or m) to $b(m)$, and by this way, the signer can not know the requester that requested the signature when the signature is submitted to the signer.

5.3 Proposed Scheme

In this section, we first describe the considered network model, then we discuss a greedy algorithm that efficiently charging the charging of the ESUs. Finally, we discuss how to implement this scheduling algorithm as a smart contract in the blockchain.

5.3.1 Network Model

As depicted in Fig. 5.1, our scheme has three main entities: Blockchain network, Energy Storage Units (ESUs), and the utility.

Blockchain network receives charging requests from ESUs and the total power load from the utility, and it computes the charging schedules in a decentralized and transparent manner. Once the charging requests and the total power load are received, the code (smart contract) that calculates the charging schedules is executed automatically and independently on each node of the blockchain network. In this work, we select Ethereum, which is the most popular open blockchain platform for smart-contracts, to implement and evaluate our proposed protocol.

Energy Storage Units (ESUs) can be electric vehicles or home batteries that communicate with the blockchain. Each ESU should acquire a list of certified pseudonyms before sending charging requests to the blockchain to anonymously authenticate the requests.

The utility posts the maximum charging capacity energy that ESUs in each community can charge on the blockchain and it does not coordinate the charging requests of ESUs.

5.3.2 Temporal Coordination of ESUs' Charging

Time is divided into a group of slots $\mathcal{T} = \{1, 2, \ldots, T\}$ with equal duration τ that covers the 24 hours of the day, and $T = 24/\tau$. We assume that each ESU belongs to a community and each community is associated with an electric bus that presents a loading limit of C kW. The regular load profile capacity within a specific community can be denoted by P_R kW at a given time slot, which accounts for residential, commercial, and industrial loads. In some cases, it is not possible that all ESUs with charging requests can be scheduled for charging at the present time slot due to the capacity limitation. Hence, a priority index is calculated for each ESU and the ESUs with the highest priorities are charged at the present time slot while ensuring that $\sum P_v \leq C - P_R$, where $C - P_R$ is the maximum available charging power and P_v is the amount of charging demand requested by an ESU. Meanwhile, other ESUs' charging requests can be deferred to future time slots.

To determine the ESU's priority for charging at the present time slot, two main components play a vital role, namely, the time to complete charging TCC (K_v) and the battery SoC (S_v). Typically, an ESU with low S_v and short K_v should have higher charging priority than an ESU with high S_v and/or long K_v. Subsequently, for each ESU v, we specify a priority index U_v such that

$$U_v = \beta_1(1 - S_v) + \beta_2 F(K_v), \tag{5.1}$$

where β_1 and β_2 are weights to determine the relative significance of S_v and K_v, with $\beta_1 + \beta_2 = 1$, $F(K_v) \in [0, 1]$ is a decreasing function and $F(K_v) = 0$ for long TCC and equals 1 for short TCC, and SoC value $(S_v) \in [0, 1)$ with $S_v = 1$ for a completely charged ESU. To illustrate, an ESU v with low S_v value and short K_v has a high priority index. Then, because of the limiting charging capacity in the present time slot, our goal is to schedule the ESUs that have the highest priority for charging in the present time slot, and postpone the charging of ESUs that have lower priorities to future time slots.

The charging schedule indicates whether a given ESU v can charge in the present time slot $(x_v = 1)$ and the amount of power (P_v) or whether it can not charge $(x_v = 0)$, where

$$\begin{aligned} \max_{x_v \in \{0,1\}} \quad & \sum_{v \in \mathcal{V}} x_v U_v \\ s.t. \quad & \sum_{v \in \mathcal{V}} x_v P_v \leq C - P_R. \end{aligned} \tag{5.2}$$

It should be noted that a scheduled ESU can obtain its entire charging demand (P_v) at the current time slot according to (5.2). To ensure an efficient resource utilization, the remaining charging capacity after scheduling ESUs according to (5.2), $C - P_R - \sum x_v P_v$, should be assigned to the ESU with highest priority among all unscheduled ESUs. Such an ESU which has not fully charged at the current time slot has to send a new charging request in the next timeslot with the remaining power demand.

The charging coordination problem in (5.2) can be well described as a Knapsack problem [178] where a set of items with different values and weights need to be packed in a knapsack of maximum capacity. The Knapsack problem aims to select items with the highest values while satisfying the knapsack capacity constraint. In this context, the ESUs are mapped to the items, the ESU priority index U_v represents the item value, the ESU charging demand P_v resembles to the item weight, and the charging limit constraint $C - P_R$ is equivalent to the knapsack limit. Hence, in order to schedule ESU charging at each time slot according to (5.2), we have modified a greedy algorithm for solving the Knapsack problem in polynomial time complexity [178]. The greedy algorithm that solves (5.2) sorts the ESU in a descending order according to the ratio between their priorities and weights. Then, ESUs with the highest orders are scheduled for charging in the current time slot, while accounting for the charging energy capacity limit. In the next section, we describe how to implement this algorithm as a smart contract in the blockchain.

5.3.3 Blockchain-based Charging Coordination

This subsection describes the proposed blockchain-based charging coordination scheme. The proposed scheme has three phases, namely, *acquiring anonymous credentials*, *charging request submission*, and *charging schedules computations*. The charging coordination algorithm described in Section 5.3.2 is implemented via a smart contract described in the pseudo code of Algorithm 5.1. The smart contract described in Algorithm 5.1 consists of a constructor named `Charging_Coordination` and the following methods (functions), namely `Recieve_Charging_Request`, `Knapsack`, and `Quick_Sort`. Two data types are supported in our smart contract, namely, address and mapping. The address is a special data type that is used to store the message caller, and the mapping resembles a hashmap that stores each ESU-related data such as TCC, SoC, and the charging power demand.

Acquiring Anonymous Credentials. In order to allow the smart contract to authenticate charging requests for a group of ESUs anonymously, each ESU should request a Partial Blind

Algorithm 5.1: Pseudo code for the charging coordination contract

```
contract Charging_Coordination
    function Charging_Coordination(_utility, C - PR)
        utility ← _utility
        MaxCapacity ← C - PR // Charging Capacity
    Struct ESU(Pv, TCC, SoC, Priority, xv, PScuduled)
    // List of ESUs sending charging requests with in a time slot
    mapping (address => ESU) ESUs
    address [] ESUlist
    // function to receive ESUs' charging requests
    function Recieve_Charging_Request(_Pv, _TCC, _SoC, TS, IDg)
        if Is_Authorized(msg.sender, PK_v^i, σ_U^{m0}(PK_v^i))
            // Verify whether the request from a legitimate ESU or not
            ESUlist.push(msg.sender)
            ESU.TCC ← _TCC
            ESU.SoC ← _SoC
            ESU.Priority ← ( ( ( 500 * 1- _TCC ) + ( 500 * _SoC ) ) / _Pv );
        end
    function Knapsack()
        // Coordinating charging requests according to algorithm discussed in Section. 5.3.2
        QuickSort(0,ESUlist.length-1);// call QuickSort
        for i ← 0 .length do
            if ESUs[ESU_list[(i)]].Pv <= MaxCapacity
                ESUs[ESU_list[i]].xv ←1
                ESUs[ESU_list[i]].PScuduled ← ESUs[ESU_list[i]].Pv
                MaxCapacity ← MaxCapacity-ESUs[ESU_list[i]].Pv
            end
        end
    function Quick_Sort(_Start, _End)
        // Sorting ESUs priority
        Start ← _Start
        End ←_End
        if Start = End  return;
        pivot ← ESUs[ESU_list[(Start + (End - Start) / 2)]].Priority;
        while Start <= End  do
            while ESUs[ESU_list[Start]].Priority > pivot)  do
                Start++
            end
            while pivot > ESUs[ESU_list[End]].Priority do
                End--
            end
            if Start <= End
                (ESU_list[Start], ESUlist[End]) ← (ESU_list[(End)], ESUlist[Start])
                Start++
                End--
            end
        end
        if _Start < End  QuickSort(_Start, End);
        if Start < _End  QuickSort(Start, _End);
```

Signature (PBS) (e.g., [180]) from the utility on public key that will be used to generate its Etherum address during charging request submission phase as follows.

Assume that the utility has a key pair (P_τ, S_τ). Each ESU v should acquire N tokens τ_i : $i = \{1, \cdots, N\}$. For each i, the ESU generates a random secret x_i and computes its public key PK_v^i, then, it blinds each PK_v^i using b_v and signs the message with its real identity (e.g., using ECDSA), and sends: $b_v(PK_v^i), \sigma_v$ to the utility, where $b_v(PK_v^i)$ is the blinded public key and σ_v

is the digital signature of the ESU v on the entire message. The utility generates a partially blind signature $\sigma_U^{m_0}(PK_v^i)$ where $m_0 = TS||ID_g$ is the appended common message, which is the current date and identifier of the group or community ID_g that the ESU belongs to. The utility then returns $\sigma_U^{m_0}(PK_v^i), \sigma_U$ to the ESU, where σ_U is the utility's digital signature on the entire message. Then, the ESU verifies σ_U, and applies the unblinding operation b_v^{-1} to obtain the token

$$\tau_i = b_v^{-1}(\sigma_U^{m_0}(b_v(PK_v^i)) = \sigma_U^{m_0}(PK_v^i),$$

and verifies that τ_i is a valid signature on PK_v^i and m_0 using the public key of the utility. Note that in this phase, although the ESU uses its real identity, the utility can not know the content of the message and this phase can occur every long period. *Charging Request Submission.* In this phase, each ESU should hide its real identity by using one of its certified public key PK_v^i along with the anonymous token τ_i, so that, the smart contract can anonymously authenticate the sender of the requests it receives. Specifically, ESU v submits a charging request $R_v = \{P_v, S_v, K_v, TS, \sigma_U^{m_0}(PK_v^i)\}, \sigma_v^i$ to the blockchain where Ts is the timestamp and σ_v is the signature of the ESU v on the entire request R_v using the SK_v^i that corresponds to PK_v^i. This signature can prove that the sender of the token knows the secret key related to the token's public key. Then, the smart contract first verifies σ_v^i and checks whether τ_i is a valid signature on PK_v^i with the appended common message m_0 using the public key of the utility P_τ. For Algorithm. 5.1, the method `Recieve_Charging_Request` can be called by an ESU to submit a charging request to the smart contract. This method calls `Is_Authorized` method that checks if the signature $\sigma_U^{m_0}(PK_v^i)$ is valid. In this case the sender's address and the request info are added to the requests list of the current time slot. Since, Ethereum does not support fixed point numbers and in order to allow our contract to compute the priority index described in Eq. 5.1, we selected selected $\beta_1 = \beta_2 = 100$ so that the priority indices are integer numbers (see line 13 in Algorithm 5.1).

Charging Schedule Computation.

By the end of each time slot, a greedy algorithm for solving the Knapsack problem is executed over the received charging requests by triggering the method `Knapsack` (see line 15 in Algorithm 5.1), which first calls the `QuickSort` method that sorts the ESUs in a descending order according to the ratio of the priority index to the amount of requested charging power. Finally, each ESU are assigned a certain charging power amount at a given time slot.

5.4 Evaluations

In this section, we evaluate the performance of the proposed scheme and compare it with First-Come-First-Serve (FCFS) approach. Also, we demonstrate a prototype for the proposed charging coordination scheme in Etherum.

5.4.1 Charging Coordination Evaluation

First, we evaluate the importance of the charging coordination by comparing it with the First-Come-First-Serve (FCFS) approach, in which the ESU that demands charging first gets charged regardless of its TCC and SoC. We conduct the experiment for 30 time slots with a battery capacity of 200 kW and maximum charging power capacity per time slot ($C - P_R$) of 1000 kW. First, there are 10 ESUs that need to charge, and a Poisson distribution with an average of λ is used to simulate the arrival process of new charging requests at each time slot. The battery SoC is a random number in the range of $[0, 1]$ that follows a uniform distribution, while TCC is a random number that follows a Geometric distribution with a average of 4. $F(K_v)$ is set to 1, 0.5, and 0, when $K_v = 1$, $K_v = 2$, and K_v ¿ 3 respectively. The results are the average of 80 runs. We use the charging index as a metric for performance evaluation. It is calculated by dividing the amount of power an ESU charges by the amount of required charging, which is a value in the range $[0, 1]$. Fig. 5.2 shows the charging index of our scheme at different values of λ compared with FCFS. It can be clearly seen

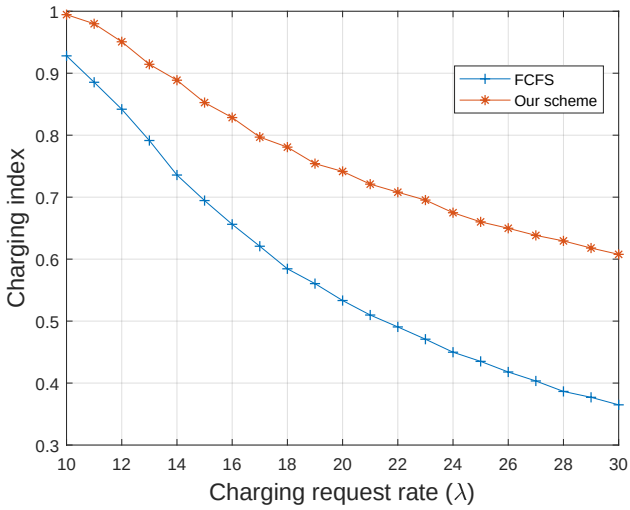


Figure 5.2: Average charging index versus charging request rate (λ).

that as the number of charging requests increases, our scheme outperforms the FCFS in serving more ESUs. This is because FCFS selects ESUs randomly while our charging scheme serves the high-priority requests first using the SoC and TCC of each ESU. Thus, more requests expire before they are served in FCFS because it does not prioritize the requests.

5.4.2 Computation Overhead

The Ethereum blockchain uses *Ethereum Virtual Machine* (EVM) to execute the code of smart contracts. This EVM is quasi-Turing complete. A machine is turing complete when it is able to solve any calculable problem given enough space and time, and it is quasi as the EVM requires enough GAS (Ether) units to operate. Every instruction consumes GAS units to be executed in the EVM. For example, to add two values from memory, 3 GAS units should be paid. If a user wants to execute a smart contract and sends a transaction to the Ethereum network that contains the instruction to do an operation, the user has to pay GAS units. We have implemented a smart contract for Algorithm 5.1 in Solidity 0.4. Then, the smart contract was deployed into the Kovan

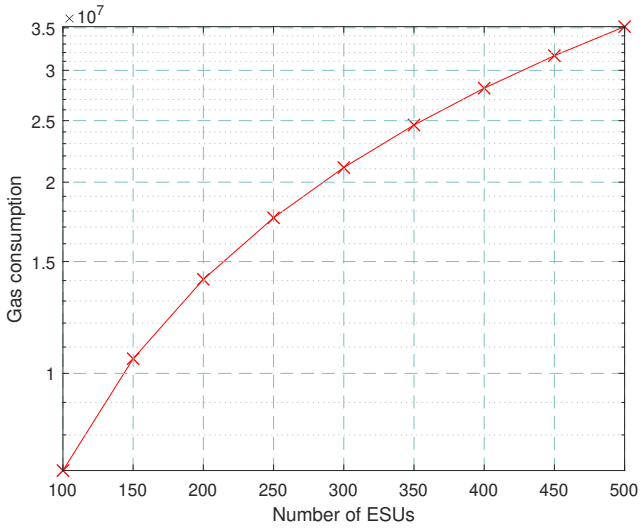


Figure 5.3: Execution cost of charging coordination contract.

blockchain [1].

The execution costs of our scheme can be considered as follows. The execution cost of deploying the smart contract in the blockchain, the cost of calling `Recieve_Charging_Request` method by an ESU, and the cost of running the `Knapsack` method to execute the charging coordination among ESUs. According to [176], the cost of 1 unit of GAS is on average 5 Gwei $= 5 \times 10^9$ ETH. Table 6.6 gives the execution cost of deploying the contract and `Recieve_Charging_Request` method. It can be noted that the costs are relatively low. For the execution cost of the charging coordination Fig. 5.3 gives the gas consumption versus the the number of ESUs. As the number of ESUs sending charging requests increases, the GAS consumption increases. Despite the reduced complexity of the proposed coordination algorithm, using the `Quick_Sort` method presents a complexity of $|\mathcal{V}| \log |\mathcal{V}|$ that causes an increase in the expected cost. While smart contracts offer a very promising way to implement protocols in a privacy-preserving and transparent manner without the need to rely on a centralized coordinator, it is still in its early evolving stage, and some limitations need to be addressed such as the scalability of number of operations made by the EVM.

[1] https://kovan-testnet.github.io/website/

Table 5.1: Summary of execution costs.

	GAS	Price (ETH)
`Deploy_Contract`	742276	0.0044537
`Recieve_Charging_Request`	76981	0.0004619

5.4.3 Security/Privacy Analysis

Our scheme presents has the following features:

1. *Decentralized charging coordination.* Since the blockchain is responsible for executing the charging coordination, our schemes does not suffer from the single point of failure and attack. An attacker needs to control a massive number of blockchain nodes in order to fail the system which is practically impossible.

2. *Privacy-preservation.* In the proposed scheme, the privacy of ESU owners (their charging requests including SoC and TCC) is protected by (*i*) replacing the ESUs' real identities by some placeholders (pseudonyms) in charging requests that correspond to temporary public-private key pairs, and (*ii*) the anonymous credentials which are obtained using PBS without allowing the utility to link them with the ESUs identities. Every pseudonym expires once the ESU sends a charging request to the blockchain which ensures *unlinkabilty* of the charging requests sent from the same ESU. Also, because of the anonymous authentication done by the blockchain, external attackers can not attack the scheme by sending charging requests while they do not belong to any ESU community.

3. *Availability.* The proposed scheme resists Denial-of-service (DoS) attacks. In such an attack, an attacker targets an ESU or even the utility to prevent legitimate transactions from appearing on the ledger, thus preventing them from posting new charging requests. To launch this attack successfully, the attacker needs to control the majority of the mining power of the network, which is practically impossible.

4. *Data integrity and transparency.* Since each ESU has access to the blockchain, it can verify the charging request sent by it and more importantly, the schedules obtained by running the modified Knapsack algorithm can be verified so that the ESU can check whether they are computed correctly. As a result, the proposed scheme offers high transparency which is not achievable in case of using centralized approaches.

CHAPTER 6

PRIVACY-PRESERVING BLOCKCHAIN-BASED ENERGY TRADING SCHEMES FOR ELECTRIC VEHICLES

In this chapter, we present our blockchain-based energy trading schemes for electric vehicles.

6.1 Overview and Main Contributions

In this chapter, by leveraging blockchain technology, we first propose a privacy-preserving CS2V scheme to enable energy trading between CSs and EVs. Then, we propose a privacy-preserving V2V energy trading scheme to enable EVs (called discharging EVs) to charge other EVs (called charging EVs). In the CS2V scheme, the CSs publish energy bids and EVs select appropriate offer and reserve it. To improve the efficiency and scalability of the scheme, we choose a *consortium* blockchain made by the charging stations to run the scheme. In the V2V scheme, we design a privacy-preserving energy trading scheme using the public blockchains which is appropriate for the V2V setting because an infrastructure to form a consortium blockchain does not exist. In this scheme, a charging vehicle sends a charging request with the region of interest and the time to find trading offers. Then, interested discharging vehicles send encrypted bids including the amount of power to sell and price to the smart contract. A charging EV, then, selects the best bid(s) and sends a reservation request to the blockchain.

Moreover, although anonymity is important to preserve the privacy of the owners of the EVs, some EVs may abuse this anonymity to launch Sybil attacks to pretend as multiple non-existing EVs and then launch powerful attacks on the system such as Denial of Service (DoS) attack. Specifically, the Sybil EVs may launch a DoS attack by making many fake reservations to block other EVs from charging thus make the energy trading system unavailable. To thwart the Sybil attacks, we

leverage a common prefix linkable anonymous authentication [151] so that a vehicle is allowed to authenticate its messages anonymously, however, if it tries to pretend as multiple EVs and submit multiple messages (reservations) with the same prefix, i.e., timeslot, the blockchain can link these submissions and know that they are sent from the same EV. However, no one can learn if two messages are sent from the same vehicle at different times to preserve privacy. In addition, we develop an anonymous and efficient blockchain-based payment system that is based on real currency and integrate it in our scheme. In particular, the system leverages a financial institution to exchange real currency with digital coins that are provably untraceable. During purchasing the coins, the financial institution can know the real identity of the buyer but when the coins are deposited by a charging station, the entity can not link coins to the buyers to preserve their location privacy. Also, coin ownership can be transferred without involving the financial institution and thus our system does not require involving the financial institution in the payment of each transaction for efficiency and scalability. In addition, our system is secure against unauthorized use of coins, by enforcing a proof of ownership for a batch coins (instead of verifying ownership of individual coins for efficiency) through a zero knowledge proof (ZKP) protocol. Moreover, our system is secure against double spending using blockchain which stores the hash of spent coins and checks for any attempt of re-spending the coins.

Our main contributions and the challenges the chapter aims to address can be summarized as follows.

- We propose energy trading schemes for CS2V and V2V that preserves the privacy of EVs owners. We also secure our system against Sybil attacks that can be used to launch powerful attacks, such as DoS, that threats the service availability.

- Moreover, an anonymous and efficient blockchain-based payment system is developed and integrated into our schemes so that the EVs owners' privacy are preserved and double-spending

and coin stolen attacks are thwarted. The payment is also efficient since the ownership of batch of coins can be verified at once through ZKP protocol.

- Extensive simulations and analysis are conducted to evaluate the proposed schemes. The results demonstrate that our schemes are secure and preserve EVs owners' privacy with acceptable communication/computation overheads.

The rest of this chapter is organized as follows. In Section 6.2, we discuss some cryptosystems that are used in our schemes. Section 6.3 discusses the considered network and threat models and design goals. Then, our proposed schemes are presented in details in Section 6.4. In Section 6.5, we present the security and privacy analysis of our schemes followed by performance evaluations.

6.2 Preliminaries

In this section, we present the necessary background needed to understand this chapter.

6.2.1 Blind Elliptic Curve DSA Signatures

The concept of blind signatures was first introduced by David Chaum in 1983 [181] as a means to provide an untraceable payment system. Using a blind signature protocol, the user (*requester*) can obtain a valid signature on a message M from the *signer* without revealing M to the signer. In our payment system, we leverage the blind signature scheme described in [182], which is based on an elliptic curve implementation of the digital signature algorithm (DSA) to create anonymous coins because it offers faster computations with significantly shorter signatures. The aforementioned protocol consists of the following steps.

1. All parties use the same elliptic curve of order n with generator G[1]. In addition, the signer's public key is $P = d \cdot G$, where d is the corresponding private key and $d \in \mathbb{Z}_n^*$.

[1]Throughout the chapter, we use uppercase letters to represent elliptic curve points, and lowercase letters to represent scalars.

2. The signer selects a uniformly random element $k \in \mathbb{Z}_n^*$ and sends $R = k{\cdot}G$ to the requester.

3. The requester selects uniformly random elements $\gamma, \delta \in \mathbb{Z}_n^*$ and computes $A = R + \gamma{\cdot}G + \delta{\cdot}P$. Let x be the x-coordinate of point A, and $t = x \bmod n$. The requester computes $c = H(M||t) \bmod n$ and sends $c' = (c - \delta) \bmod n$ to the signer. $H(\cdot)$ is a cryptographically secure hash function, such as SHA-256.

4. The signer computes $s' = (k - c'{\cdot}d) \bmod n$ and sends the result back to the requester.

5. The requester computes $s = (s' + \gamma) \bmod n$ and stores the signature of M as (s, c).

6. To verify the signature, the *verifier* computes $A = c{\cdot}P + s{\cdot}G$. Then, it computes $t = x \bmod n$, where x is the x-coordinate of point A. The verifier checks if $c \stackrel{?}{=} H(M||t) \bmod n$.

6.2.2 Schnorr's Identification Protocol

An identification protocol allows the owner of a public key (*prover*) to prove to a *verifier*, in zero knowledge, that he indeed knows the value of the underlying secret key. A well-known identification protocol is called Schnorr [183], which is summarized below for the case of an elliptic curve cryptosystem.

1. We assume that all parties use the same elliptic curve of order n with generator G. The prover's public key is $P = d{\cdot}G$, where d is the corresponding private key.

2. The prover selects a uniformly random element $k \in \mathbb{Z}_n^*$ and sends $R{=}k{\cdot}G$ to the verifier (*commitment*).

3. The verifier selects a random element $e \in \mathbb{Z}_n^*$ and sends it to the prover (*challenge*).

4. The prover computes $s = (k + e{\cdot}d) \bmod n$ and sends it to the verifier (*response*). The verifier accepts the proof if $s{\cdot}G = R + e{\cdot}P$.

Gennaro et al. [184] use higher degree polynomials that enable the execution of multiple Schnorr protocol instances simultaneously with overhead that is very close to the overhead of a single instance. The protocol is identical to the one described above, except for the last step. In particular, the prover holds m public keys $P_1, P_2, \ldots, P_m$, corresponding to private keys $d_1, d_2, \ldots, d_m$. When the prover receives the verifier's challenge, it computes the response $s = (k + \sum_{i=1}^{m} e^i \cdot d_i) \bmod n$. The verifier then accepts the proof by checking if $s \cdot G \stackrel{?}{=} R + \sum_{i=1}^{m} e^i \cdot P_i$

In our schemes, we uses the batch version of Schnorr's protocol to allow the owner of the coins, i.e., EV/CS (prover) to deposit the digital coins into their account by proving the ownership of their digital coins to the financial institution (verifier). Also, the scheme can be extended to allow the the verifier to verify proves of two provers at once by simply checking if $(s_1 + s_2) \cdot G \stackrel{?}{=} R_1 + R_2 + \sum_{i=1}^{m} e^i \cdot P_i + \sum_{j=1}^{m} e^j \cdot P_j$, where ($R_1$, s_1) and (R_2 and s_2) are the commitment and response values from two provers, respectively.

6.2.3 Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK)

A zero-knowledge proof (zk-proof) allows a party (i.e., prover) to generate a cryptographic proof convincing another party (i.e., verifier) that some values are obtained by faithfully executing a pre-defined operations on some private inputs (i.e., witness) without revealing any information about the private state [152]. The zk-SNARK further allows such a proof to be generated non-interactively. More importantly, the proof is *succinct*, i.e., the proof size is independent on the complexity of the statement to be proved. More precisely, zk-SNARK has three phases. A setup phase outputs the public parameters to establish a SNARK for a NP-complete language L. In the proof generation phase, the *Prover* uses an instance x, and witness w to generate a proof for the statement $x \in L$. Finally, in the verification phase, the *Verifier* can efficiently verifies the proof.

6.2.4 Common-prefix-linkable anonymous authentication

Recenlty, Lu et. al, [151] have proposed a common prefix linkable anonymous authentication scheme based on zk-SNARK descried in Section. 6.2.3. In this scheme, a user can authenticate messages anonymously and prove the validity of his certificate without being identified. The only exception is when the same key holder authenticates two messages with the same prefix. In this case, everyone can link the authentications. The scheme consists of the following five algorithms:

- $Setup(1^\lambda) \rightarrow (PP, msk, mpk)$: This algorithm establishes the public parameters PP needed for the zk-SNARK system and the system's master public key mpk, and system's master secret key msk.

- $CertGen(msk, PK_i) \rightarrow cert_i$: This algorithm outputs certificate $cert_i$ to validate the public key PK_i.

- $Auth(p||m, SK_i, PK_i, cert_i, PP) \rightarrow \pi = (t_1, t_2, \eta)$: This algorithm generates an attestation π on a message m and a prefix p that the sender indeed owns a secret key corresponding to a valid certificate $cert_i$. The algorithm first computes two tags, $t_1 = H(p, SK_i)$ and $t_2 = H(p||m, SK_i)$, where H is a secure hash function, e.g., SHA-256. Then, let $\vec{w} = (SK_i, PK_i, cert_i)$ represents the private witness, and $\vec{x} = (p||m, mpk)$ be all common knowledge, the algorithm runs zk-SNARK proving algorithm $Prover(\vec{x}, \vec{w}, PP)$ for the following language $_T = \{t_1, t_2, \vec{x} = (p||m, mpk) \mid \exists \vec{w} = (SK_i, PK_i, cert_i)$ s.t. $CertVrfy(cert_i, PK_i, mpk) = 1 \wedge pair(PK_i, SK_i) = 1 \wedge t_1 = H(p, SK_i) \wedge t_2 = H(p||m, SK_i)\}$, where the $CertVrfy$ algorithm checks the validity of the certificate using a signature verification, and $pair$ algorithm verifies whether two keys are a public-secret key pair. Finally, the algorithm outputs $\pi = (t_1, t_2, \eta)$.

- *Verify*$(p||m, \pi, mpk, PP) \rightarrow \{0,1\}$: this algorithm runs the zk-SNARK *Verifier* algorithm on $\vec{x}, \pi$ and PP, and outputs 0/1 to decide whether the attestation is valid or not for the attested message.

- *Link*$(m_1, \pi_1, m_2, \pi_2) \rightarrow \{0,1\}$: On inputting two attestations $\pi_1 = (t_1^1, t_2^1, \eta_1)$ and $\pi_2 = (t_1^2, t_2^2, \eta_2)$, the algorithm simply checks $t_1^1 \stackrel{?}{=} t_1^2$. If the check is true, the algorithm outputs 1; otherwise, output 0. We also use *Link*(π_1, π_2) for short. Obviously, this operation is very efficient because it does not need cryptographic operations.

6.3 Network/Threat Models and Design Goals

In this section, we present the considered network model followed by the adversary and threat models, and then, we introduce the design goals of our schemes.

6.3.1 Network Models

Figs. 6.1 and 6.2 illustrate the network models of the CS2V and V2V schemes, respectively. The main entities are charging stations, charging EVs, discharging EVs and the Financial institution (FI). In addition to these entities that are involved in running our schemes, there are also an offline Key Distribution Center (KDC). The KDC's role is to issue public key certificates to the charging stations and EVs that participate in the energy trading and the financial Institution that runs the anonymous payment system. In practice, the KDC can be run by a governmental agency which is interested in the security of the energy trading system [185]. The FI sells *untraceable* digital coins to EVs and exchanges these coins for real currency.

As shown in Fig. 6.1, in the CS2V scheme, there are charging station(s) owned by private companies that charge EVs. The charging stations can have multiple points that can charge many EVs simultaneously or they can be a few charging points installed in parking lots of clinic, shopping

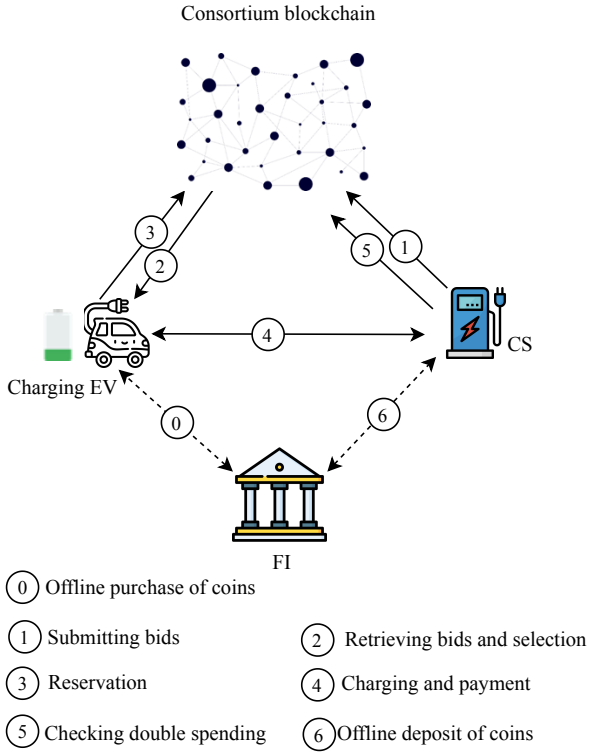


Figure 6.1: Network model of the CS2V scheme.

malls, work places, etc. Each EV interacts with the system through a Web or mobile application. The blockchain network is a consortium network made by the charging stations that run our scheme by processing all transactions' messages sent by the different entities.

As shown in Fig. 6.2, in the V2V scheme, EVs can be charging vehicles which seek to buy energy or discharging vehicles which have excessive energy to sell. Unlike the CS2V scheme that uses private bkockchain, public blockchains, such as Ethereum [2], are used in the V2V scheme. This is because in the V2V scheme, there is no available charging stations or other dedicated entities that can form a consortium blockchain.

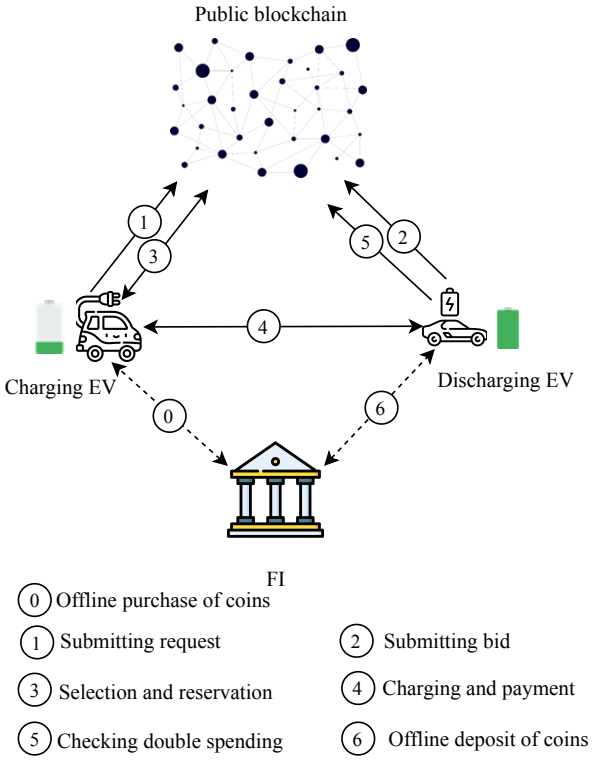


Figure 6.2: Network model of the V2V scheme.

6.3.2 Adversary and Threat Model

Security threats in the energy trading system can come from both internal and external adversaries [186]. In this chapter, we assume that the entities (FI, CSs, EVs, and validators/miners) are honest-but-curious, i.e., they execute our schemes correctly, but aim to infer sensitive information about the EVs' drivers by examining the exchanged messages/transactions. We also consider external adversaries that have access to the blockchain and may try to impersonate legitimate users. These adversaries can eavesdrop on all the exchanged messages and all previous recorded transactions on the blockchain for EVs' trading transactions so that they try to learn the visited locations and the driving patterns, guess the locations of drivers at a specific time or even track them over time. In addition, attackers may launch Sybil attacks to pretend as multiple different non-existing EVs to launch severe attacks such as DoS attacks by submitting many reservations to deprive hon-

est EVs from getting charged or selling their energy which makes the service unavailable. We also consider attacks against the payment system. Specifically, some attackers may try to create fake coins and double-spend valid coins. Finally, we assume that all parties devices run in polynomial time and are, thus, unable to break the cryptographic protocols.

6.3.3 Design Goals

Our schemes should achieve the following important objectives.

- Resilience/security. The proposed schemes should not rely on a central entity to run the energy trading system.
- *Privacy Preservation.* The EV owners' privacy is protected by achieving the following two requirements. (*i*) Anonymity: anyone including the CS, blockchain, and FI should not be able to identify the real identity of an EV that sends a message or a transaction in the energy trading system. (*ii*) Unlinkability: nobody can link two bid/offer/reservation messages sent from the same EV at different times.
- *Anonymity-yet-resistance to Sybil attack.* Launching Sybil attacks by pretending as multiple non-exiting EVs and launch severe attacks such as DoS by submitting a large number of offers/requests pretending that they are submitted from different entities should be thwarted.
- *Authentication.* Electric vehicles should be authenticated in an anonymous way so that no adversary can impersonate a legitimate EV.
- *Efficient anonymous payment system.* The payment scheme should be anonymous and secure against fake payments and double spending attacks. It should also be scalable and efficient.

6.4 Proposed Privacy-Preserving CS2V and V2V Energy Trading Schemes

In this section, we describe in detail our privacy-preserving energy trading schemes. We first describe the system initialization phase, followed by the purchase of digital coins phase. Then, we describe in details the CS2V scheme. Finally, we describe the V2V scheme.

6.4.1 System Initialization

In this phase, the KDC issues public key certificates to the charging stations, EVs, and the financial institution involved in the anonymous payment system. The KDC first generates a master public/private key pair (mpk/msk) which is used for a digital signature scheme, susch as RSA, for signing the certificates. Then, an EV v_i, having a unique identity (e.g., license plate number), creates a public/private key pair (PK_{v_i}/SK_{v_i}) and KDC generates a certificate $cert_{v_i}$ binding PK_{v_i} to v_i. The KDC also runs the *Setup* algorithm of zk-SNARK to generate the public parameters (PP) of the zk-SNARK. Finally, the KDC publishes the zk-SNARK parameters PP and the master public key mpk to the system users. These parameters are used by the EVs to authenticate messages using the common-prefix linkable anonymous authentication. Note that this phase is done off-line and only once. Finally, we assume that all parties use the same elliptic curve group of order n with generator G.

6.4.2 Purchasing Digital Coins

In our schemes, EVs need to purchase untraceable coins from the FI. This process is performed *outside* the blockchain. Specifically, we assume that the energy trading application on the EV driver's smart phone contains an *e-wallet* service, which stores coins that are digitally signed by the FI. The FI only issues coins with predefined momentary values so that the coin value can not be used to identify a specific EV because many EVs buy coins with the same value. The purchasing of digital coins proceeds as follows.

1. Suppose the EV (client) wants to purchase m digital coins from the FI. After the payment is made (e.g., through a credit card), the client chooses m secret and random *elements* $s_1, s_2, \ldots, s_m \in \mathbb{Z}_n^*$ and computes m coin public keys $CP_i = s_i \cdot G$, $1 \leq i \leq m$. Each public key uniquely identifies one coin.

2. The client and the FI invoke the blind signature protocol discussed in Section 6.2, so that the client obtains a valid signature $sig_{FI}(CP_i)$ for each purchased coin CP_i as follows.

   Step.1: the EV sends a purchase request message msg_1

   $$msg_1 = ID_v \| m \| Sig_v(ID_v \| m)$$

   where ID_v is the EV real identity and m is the number of digital coins and their momentary values and $Sig_v(ID_v \| m)$ is the signature on the whole message using the EV private key.

   Step.2: the FI chooses m secret and random *elements* $k_1, k_2, \ldots, k_m \in \mathbb{Z}_n^*$ and computes $R_i = k_i \cdot G$, $1 \leq i \leq m$. Then, it sends msg_2 to the EV

   $$msg_2 = R_i \| Sig_{FI}(R_i),\ \forall i \in \{1, 2, ..., m\}$$

   Step.3: the EV computes $A_i = R_i + \gamma_i \cdot G + \delta_i \cdot P$, $\forall i \in \{1, m\}$, where $\gamma_i, \delta_i \in \mathbb{Z}_n^*$ are random elements and computes $t_i = x_i \bmod n$, where x_i is the x-coordinate of point A_i. The EV computes $c_i' = (c_i - \delta_i) \bmod n$, where $c_i = H(CP_i || t_i) \bmod n$. The EV sends msg_3 to the FI

   $$msg_3 = c_i',\ \forall i \in \{1, 2, ..., m\}$$

   Step.4: The FI computes $s_i' = (k_i$ - $c_i' \cdot d)$ mod n where d is the FI secret key and sends msg_4

   $$msg_4 = s_i',\ \forall i \in \{1, 2, ..., m\}$$

to the EV.

Finally, the EV computes $s_i = (s'_i + \gamma_i) \bmod n$ and stores the signature on CP_i ($Sig_{FI}(CP_i)$) as (s_i, c_i). The FI remains oblivious to the values (i.e., public keys) of the individual coins. Note that in this phase the client uses his real identity to purchase coins and because of using the blind signatures, the FI can not link a coin to the EV that bought it when the coin is deposited by a CS and this is required to preserve privacy.

3. To improve the efficiency of our system, we can allow digital coins of different denominations (e.g., \$1, \$5, \$10). In this case, the FI signs each denomination with a different private key dedicated to this denomination, and each coin is stored as $\langle v, CP_i, sig_{FI^{(v)}}(CP_i)\rangle$, where v is the coin's monetary value, $sig_{FI^{(v)}}(CP_i)$ is the signature of the FI on CP_i using its private key dedicated for the v value. This approach is not prone to privacy leaks by linking a coin value to the client who bought it because coins with the same domination value are bought by many EVs.

6.4.3 Privacy-Preserving CS2V Energy Trading Scheme

As illustrated in Fig. 6.1, the CS2V scheme consists of the following phases. In the *submitting bids* phase, CSs submit competitive charging bids to the blockchain. In the *reservation* phase, an EV that needs to charge submits a reservation request to the blockchain, and the validators verify the reservation request and store it in the blockchain. In the *charging and payment* phase, the EV charges and pays. Finally, the last phase is the *coin deposit* where the CS deposits the coins in the FI. The exchanged messages in CS2V scheme are shown in Fig. 6.3.

Submitting Bids. In this phase, the charging stations publish their energy bids on the blockchain as follows. First, the day is divided into predefined timeslots, e.g., each timeslot can

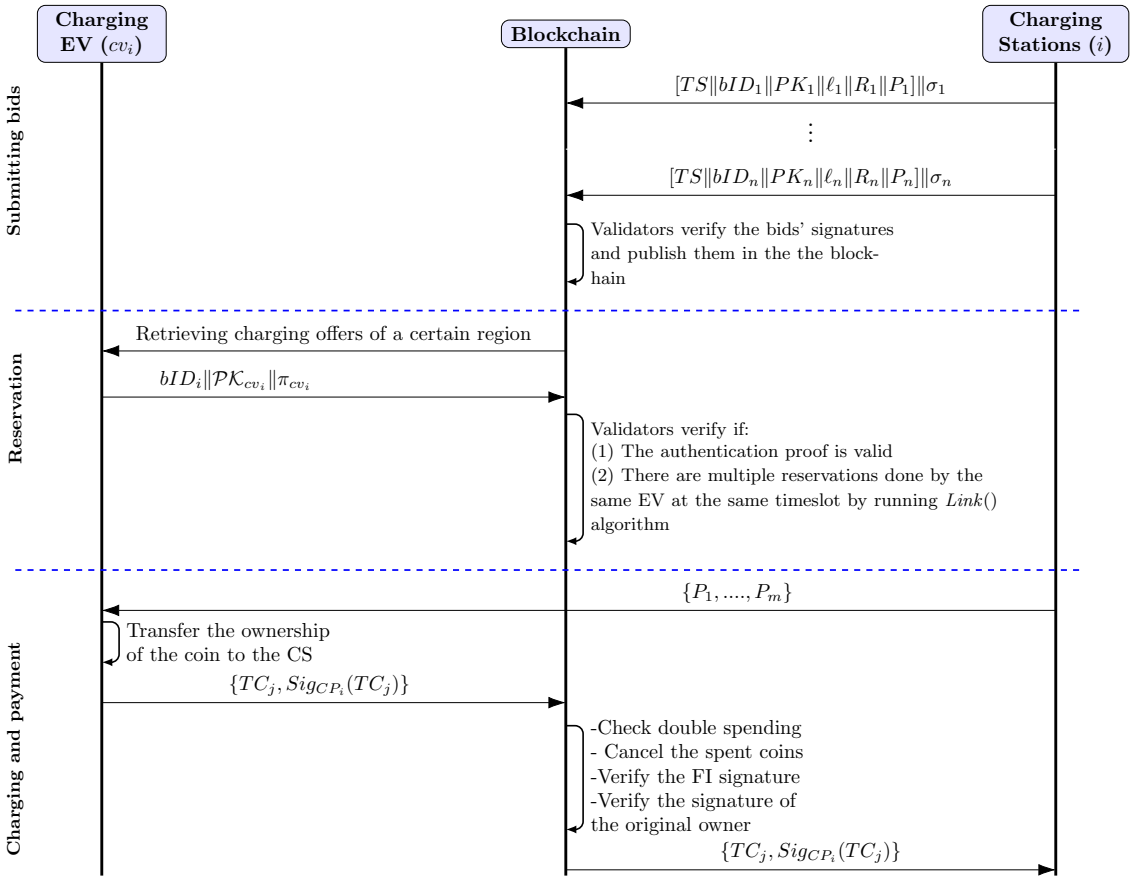


Figure 6.3: Illustration of the exchanged messages in our CS2V energy trading scheme.

be 30 minutes or 1 hour. Then, when a CS i has an available charging point, it composes a *bidding* message msg_5:

$$msg_5 = TS\|bID_i\|PK_i\|\ell_i\|R_i\|P_i,$$

where TS is the timeslot of charging, bID_i is bid ID, PK_i is CS's public key, ℓ_i is location of the CS, R_i is the charging rate (price of each KWh), and P_i is the charging energy (KWh) it can provide.

The message is signed with the private key of the CS and is broadcasted on the blockchain network. Note that a charging station can send multiple bids at a specific timeslot and the number

of bids depends on the number of EVs the charging station can charge at a certain timeslot. Before storing the bid on the shared ledger, the validators of the blockchain network should verify the signature of the message.

Reservation. In this phase, a charging EV cv_i that needs to charge queries the blockchain to retrieve the bids of the charging stations of a certain geographic area. Then, the EV should select the most appropriate bids(s), in terms of distance to the CS, charging rate, and the amount of energy. The selection of best bid(s) depends on the preference of the charging vehicle. For instance, an EV may prefer a nearby charging station regardless of the charging rate while others may prefer lower charging rates. Note that an EV can select multiple bids at different timeslots in case it needs to schedule charging for a long time, e.g., while driving the EV for a long trip, or when it does not find one bid that satisfies its energy need. In this section, we explain an approach to enable EVs to select the most appropriate bid. Three main factors play a key role in selecting best bids, namely, the amount of energy of the bid (P_j), the distance to the charging station (D_j) and the charging rate (R_j). Typically, a CS's bid with high P_j, short D_j and low R_j should have high priority. Subsequently, for each bid j, the EV calculates its priority index U_j as follows.

$$U_j = \beta_1 F_1(P_j) + \beta_2 F_2(D_j) + \beta_3 F_3(R_j) \tag{6.1}$$

where β_1, β_2, and β_3 are weights that determine the relative significance of P_j, D_j and R_j, where $\beta_1 + \beta_2 + \beta_3 = 1$. $F_1(P_i)$ is an increasing function of the amount of energy P_j with a range of $[0, 1]$ where $P_j = 1$ if the bid can fully satisfy the charging need of the charging EV. Both $F_2(D_j)$ and $F_3(R_j)$ are decreasing functions of D_j and R_j, respectively, with a range of $[0, 1]$, where $F_2(D_j) = 0$ for a long distance while $F_2(D_j) = 1$ for a short distance, and $F_3(R_j) = 0$ for high charging rate while $F_3(R_j) = 1$ for low charging rate. Then, our goal is to enable cv_i to select the bid(s) that have the highest priorities and give the amount of energy it needs. We formulate this problem using

Eq. 6.2 where the decision indicates whether a given bid is selected ($x_j = 1$) with the energy to charge ($P_j = 1$) or is not selected ($x_j = 0$), and P_{cv_i} is amount of power cv_i wants to charge

$$\begin{aligned} &\max_{x_j \in \{0,1\}} && \sum_{\forall j \in \{1,2,\dots\}} x_j U_j \\ &s.t. && \sum_{\forall j \in \{1,2,\dots\}} x_j P_j \leq P_{cv_i} \end{aligned} \tag{6.2}$$

Solving Eq. 6.2 can be well described as a Knapsack problem [178] where a set of items with different values and weights need to be packed in a knapsack of maximum capacity. The Knapsack problem aims to select the items with the highest values without exceeding the knapsack capacity constraint. In this context, the bids are mapped to items, the priority U_j represents the item value, P_j is mapped to the item weight, and the demanded energy P_{cv_i} is equivalent to the knapsack's maximum capacity. A greedy algorithm for solving the Knapsack problem [178] in polynomial time complexity as illustrated in Algorithm 6.1 can be used to select the best bids. Note that before running the algorithm, we should make sure that there is no conflict among the bids inputted to the algorithm, i.e., bids that can not be selected at the same timeslot, e.g., because they are in the same timeslot or the charging stations are from each other. This may necessitate running the algorithm multiple times with different sets of non-conflicting bids which may produce different possible solutions so in this case the EV should select the best solution based on its priority index and/or amount of power that can be charged.

Once the cv_i determines the best bid(s), it then creates a new public/private key ($\mathcal{PK}_{cv_i}/\mathcal{SK}_{cv_i}$) used for once and the corresponding address $\mathcal{AD}_{cv_i}$. Then, it uses common-prefix-linkable anonymous authentication scheme to generate an attestation π_{cv_i} by running $Auth$ algorithm descried in Section 6.2.4 and using the zk-SNARK's public parameters PP and the timeslot TS included in the bid as the prefix as follows:

Algorithm 6.1: Selection of the best bids.

Input: Set of bids and P_{cv_i};
Initialization: $\mathcal{A} = \{\}$, $C_R = P_{cv_i}$;
Calculate each bid priority U_j using Eq. 6.1
Divide each bid priority index by the amount of power it can sell, then sort the results in increasing order and store them in $\mathcal{A}$;
$x[\,]$: empty array for selected bids
$CP[\,]$: an array of charging power
for $j \in \mathcal{A}$ do
  if $P_j \leq P_{cv_i}$ then
    $x[j] = 1$;
    $C_R = C_R - P_j$;
    $CP[j] = P_j$;
    $\mathcal{A} = \mathcal{A} - \backslash\{j\}$;
  end if
end for
Output: Selected bid(s) $\{x[j], CP[j]\}$.

$$\pi_{cv_i} = Auth(TS||\mathcal{AD}_{cv_i}, PK_{cv_i}, SK_{cv_i}, cert_{cv_i}, PP),$$

Then, cv_i creates a *reservation* message msg_6:

$$msg_6 = bID_j \| \mathcal{PK}_{cv_i} \| \pi_{cv_i}$$

where bID_j is the selected bid ID, π_{cv_i} is the authentication proof and $\mathcal{PK}_{cv_i} = k.G$ is the one-time public key computed by the cv_i where $k \in \mathbb{Z}_n^*$. Then, the message is broadcasted on the blockchain network. Note that $\mathcal{PK}_{cv_i}$ will be used later by the cv_i to prove to the CS that it is the one that indeed made the reservation.

Then, an EV reservation request is verified and stored in the blockchain. To do so, the validators check that the attestation proof π_{cv_i} is valid by running *Verify* algorithm of common-prefix-linkable anonymous authentication scheme described in Section 6.2.4 as follows:

$$Verify(TS||\mathcal{AD}_{cv_i}, \pi_{cv_i}, mpk, PP),$$

Also, the validators check for multiple reservations by executing $Link(\pi_{cv_i}, \pi_{cv_*})$ discussed in Section 6.2.4 for each valid authentication attestation π_{cv_*} that was received in the timeslot TS. Once all these verifications succeed, the reservation transaction is stored on the ledger. Note that running the *Link()* algorithm is very efficient because it does not require cryptographic operations from the validators and it is just checking the equality of the two authentication attestations as described in Section 6.2.4, and this makes our scheme scalable.

EV Charging and Payment. After reserving charging, in this phase, an EV charges from a charging station and pays. First, when the EV arrives at the charging station, it has to prove that it has reserved a charging point in the CS. In particular, the EV and the CS engage in an instance of Schnorr's identification protocol (Section 6.2), in order for the EV to prove to the CS that it knows the private key corresponding to the public key that made the reservation. Following a successful authentication, the EV charges the amount of power in the reservation and then initiates the payment procedure.

The CS chooses m secret random *elements* $k_1, k_2, \ldots, k_m \in \mathbb{Z}_n^*$ and computes m public keys $P_j = k_j \cdot G, 1 \leq j \leq m$ for each coin. The EV transfer the ownership of the coin to the CS as follows.

The EV first selects a random element $a \in \mathbb{Z}_n^*$, then it computes $a.G$, $r = x_1 \bmod n$, where x_1 is the x-coordinate of the point $a \cdot G$. Then, it computes $a^{-1} \bmod n$ and $B = H(TC_j)$, where $TC_j = CP_i \| sig_{FI}(CP_i) \| P_j$. Then, it uses the secret key s_i of the coin $CP_i \| sig_{FI}(CP_i)$ to compute $S = a^{-1}(B + s_i \cdot r) \bmod n$. The value (S, r) is the signature $Sig_{CP_i}(TC_j)$ on (TC_j). Thus, $TC_j \| Sig_{CP_i}(TC_j)$ is the new transferred coin. Then, the EV sends the transferred coins to the blockchain. The blockchain should do the following: (i) verify the FI signature, (ii) check if the coins have not been spent before by checking the list of spent coins on the blockchain and if not stores and (iii) verify the signature of the transferred coins as follows. it Computes $B = H(TC_j)$, $u_1 = B.S^{-1} \bmod n$, and $u_2 = r.S^{-1} \bmod n$. Then, it computes $(x_1, y_1) = u_1.G + U_2.CP_j$, if $x_1 = r$ then the signature is valid. If all the verifications succeeds, the old coins are stored in the list of

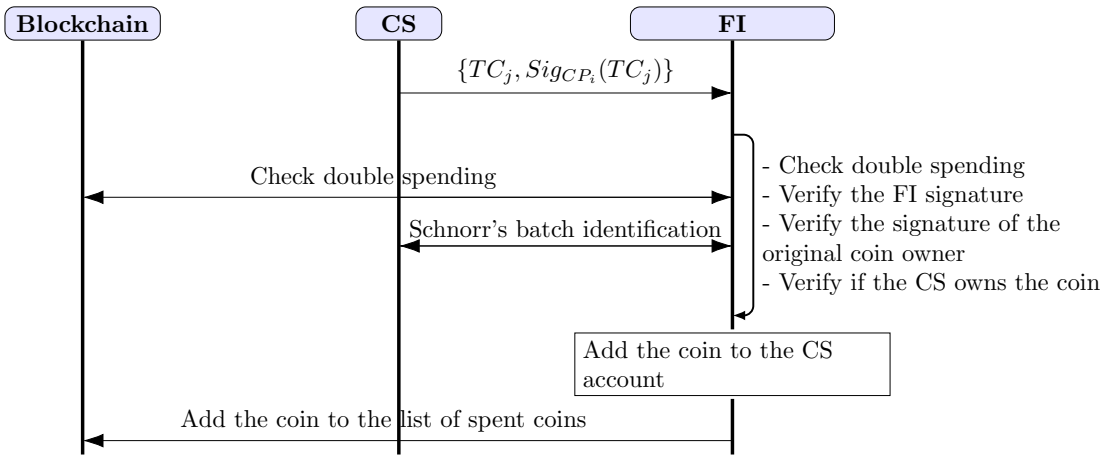


Figure 6.4: Coin deposit.

spent coins and the blockchain send transferred coins to the CS.

Coin Deposit. In this phase, the CS deposits the coins it collects in a period of time (e.g., several days) in the FI. Fig. 6.4 illustrates the exchanged messages in this phase. The CS sends a batch of coins to the FI. The FI verifies the following: (*i*) the coins have not been spent before by checking if the hash of the coin in the list of spent coins, (*ii*) the FI signature of the coin is valid, (*iii*) the signature of the old owner is valid and (*iv*) finally, checks if the CS owns the coins are not; to do so, the CS and the FI should invoke the batch version of Schnorr's protocol described in Section 6.2.2 so that the CS can prove that it knows the underlying secrets of the submitted coins. If all the verifications succeed, the FI deposits the coins to the CS account and the CS can exchange coins with real currency if it wants. Note that the FI sends the exchanged coins to the blockchain to be stored in the list of spent coins to avoid re-depositing the same coin.

6.4.4 Privacy-Preserving V2V Energy Trading Scheme

As illustrated in Fig. 6.2, the V2V scheme consists of the following phases. In the *charging request* phase, a charging EV submits a charging request to the blockchain containing cloaked location, time of charging, and anonymous signature. In the *charging bids* phase, the discharging

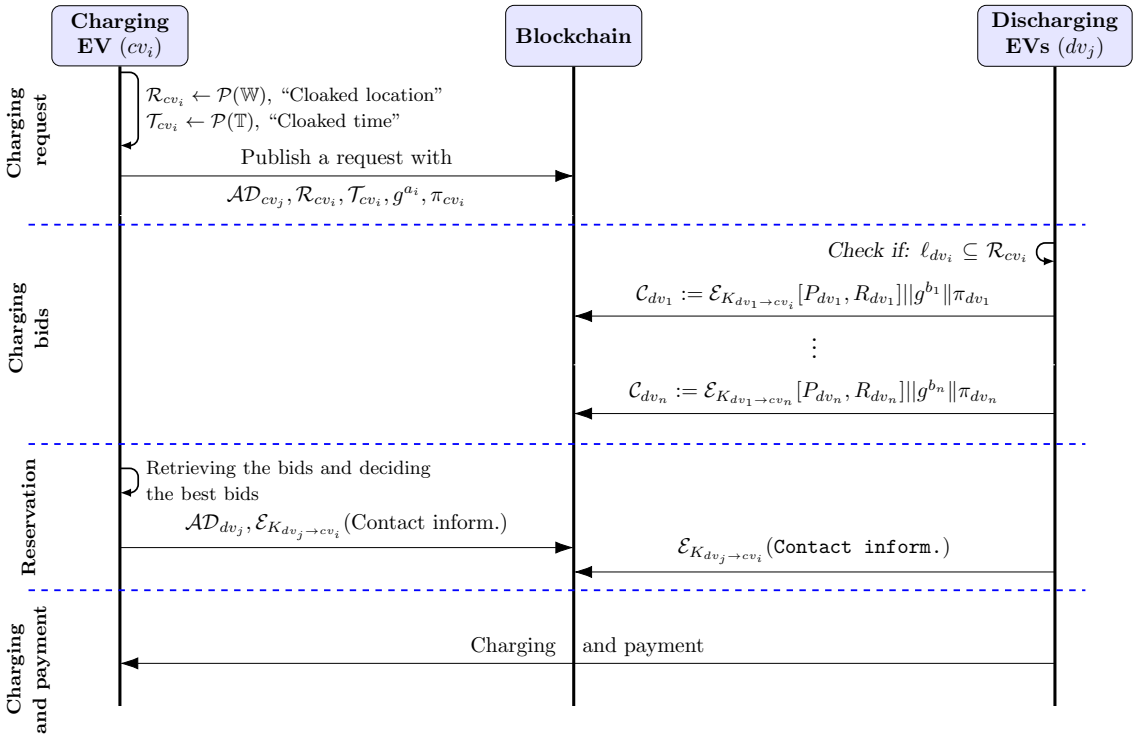


Figure 6.5: The V2V energy trading scheme. Note that $\mathcal{E}$ denotes a symmetric key encryption algorithm e.g., AES-128.

EVs submit authenticated and anonymous bids to the blockchain. In the *reservation* phase, the charging vehicle selects the best bids and submits a deposit to the selected discharging EVs. Finally, in the *charging and payment* phase, the EV charges and pays. Finally, the last step is the *coin deposit* where the discharging EV deposits the coins in the FI. An illustration to the exchanged messages in the V2V scheme are shown in Fig. 6.5.

Charging request. In this phase, a charging vehicle cv_i submits an anonymous authenticated charging request to the blockchain so that the discharging EVs submit their bids. The smart contract given in Algorithm 6.2 is responsible for executing the V2V energy trading scheme by receiving charging requests from charging EVs and bids from discharging EVs.

To preserve location privacy, let the area $\mathcal{A}$ (e.g., a city) where energy is traded is divided into

a set of regions $\mathcal{R}$ (smaller geographic areas). Then, cv_i composes an energy request by selecting a cloaked location, i.e., geographic region $\mathcal{R}_{cv_i}$, instead of exact location. It also cloaks the desired time of charging $\mathcal{T}_{cv_i} \in \mathcal{P}(\mathsf{T})$ where $\mathcal{P}(\mathsf{T})$ is a set of time intervals.

Then, cv_i uses common-prefix-linkable anonymous authentication scheme to generate an attestation π_{cv_i} by running $Auth$ algorithm using the zk-SNARK's public parameters PP and the timeslot (TS) as the prefix as follows;

$$\pi_{cv_i} = Auth(TS||\mathcal{AD}_{cv_i}, PK_{cv_i}, SK_{cv_i}, cert_{cv_i}, PP),$$

Next, cv_i chooses a random value $a_i \in \mathbb{Z}_n^*$ and computes g^{a_i}. The cv_i sends the tuple $\mathcal{AD}_{cv_i}$, $\mathcal{R}_{cv_i}$, $\mathcal{T}_{cv_i}$, g^{a_i} and π_{cv_i} to the smart contract given in Algorithm 6.2. Then, the contract $\mathcal{C}$ verifies the proof π_{cv_i} (see line 6-10 in Algorithm 6.2) by running:

$$Verify(TS||\mathcal{AD}_{cv_i}, \pi_{cv_i}, mpk, PP).$$

We should also note that $\mathcal{C}$ ensures that each charging EV can not submit multiple requests at the same timeslot by executing $Link(\pi_{cv_i}, \pi_{cv_*})$ for each valid authentication attestation π_{cv_*} that was received and stored in the smart contract at a specific timeslot (see line 5-11 in Algorithm 6.2).

Charging Bids. In this phase, discharging vehicles interested in a certain energy request send their authenticated and encrypted offers to the smart contract. Then, dv_j checks available charging requests on the blockchain and checks whether its location ℓ_{dv_j} lies in the region of the request $\mathcal{R}_{cv_i}$. Then, it chooses a random value $b_j \in \mathbb{Z}_n^*$ and computes g^{b_j}. Then, dv_j uses g^{a_i} included in the trading request to compute a symmetric key $k_{dv_j \to cv_i}$ as follows:

$$k_{dv_j \to cv_i} = (g^{a_i})^{b_j} = g^{a_i b_j}$$

Algorithm 6.2: Pseudocode of the V2V energy trading contract

```
contract EnergyTrading
    PP // SNARKs public parameters and KDC's master public key
    mapping(address => string) Proofs // Mapping for received attestations
    mapping(address => string) Reserved DVs // Mapping for reserved discharging
       EVs
    function EnergyTrading(R_{cv_i}, T_{cv_i}, π_{cv_i}, g^{a_i} )
    if Verify(TS||AD_{cv_i}, π_{cv_i}, mpk, PP)!=1
        return // Unauthenticated request
    else
        ∀ π_* ∈ Proofs ! Link(π_{cv_j}, π_*))
        Proofs[]← π_{cv_i} ;
    end
    function RecieveOffer((C_{dv_j}, g^{b_j}, π_{dv_j}, AD_{cv_i}))
        address [] DVlist // Received discharging EVs' offers
        address [] DVProofs // Received discharging EVs' proofs
        address [] bids // Received bids
        if Verify(AD_{cv_i}||AD_{dv_j}, π_{dv_j}, mpk, PP)!=1
            return // Unauthenticated offer
        else
            ∀ π_* ∈ Reserved DVs ! Link(π_{dv_j}, π_*))
            return // Reserved discharging EV
        end
    function Reservation(AD_{dv_j})
        receive the selected addresses of discharging vehicles.
        Reserved DVs[]← π_{dv_j} ;
```

Then, it encrypts the tuple (P_{dv_j}, R_{dv_j}) with $k_{dv_j \to cv_i}$ using a symmetric key encryption algorithm, e.g., AES-128, to obtain the ciphertext C_{dv_j}, where P_{dv_j} is the amount of energy that dv_j can sell and R_{dv_j} is the charging rate (price/KW). Note that we use symmetric key encryption since it is much efficient than asymmetric key encryption in terms of computation and computation overheads. Then, dv_j uses common-prefix-linkable anonymous authentication scheme to generate an attestation $\pi^1_{dv_j}$ by running $Auth$ algorithm and TS as the prefix as follows:

$$\pi_{dv_j} = Auth(TS||\mathcal{AD}_{dv_j}, PK_{dv_j}, SK_{dv_j}, cert_{dv_j}, PP),$$

Then, it sends $\mathcal{C}_{dv_1} = \mathcal{E}_{K_{dv_1 \to cv_i}}[P_{dv_1}, R_{dv_1}]||g^{b_1}||\pi_{dv_1}$ to the smart contact. Note that π_{dv_j} is used so that the contract can check if the the discharging vehicle dv_j has not already been reserved

in the timeslot more than once. Also, discharging EVs are allowed to send multiple bids in the same timeslots for different requests because it not guaranteed that it will be selected by other charging EVs, however the contract prevent reserving the same discharging EV for more than one charging EV (see line 19 in Algorithm 6.2)

Reservation. In the reservation phase, once the smart contract collects the bids from the discharging vehicles, the charging EV decides the best bid(s) and makes the reservation. To do that, cv_i first retrieves the bids from the smart contract. Then, it computes a symmetric key shared with each dv_j that has sent a bid. For example, cv_i uses g^{b_j} included in the bid of dv_j to compute a symmetric key $k_{cv_i \to dv_j}$ as follows:

$$k_{cv_i \to dv_j} = (g^{b_j})^{a_i} = g^{b_j a_i}$$

Then, cv_i decrypts all the bids and selects the best bid(s). One way to select best bid(s) is using the procedure explained in subsection **??**. Finally, upon determining the best bid(s), cv_i sends a transaction to the contract that contains the IDs of the selected discharging EVs and the contact information, i.e., email address or phone number of the charging EV, encrypted with the shared secret key. Finally, the selected discharging vehicle sends its contact information encrypted by the shared symmetric key with the charging vehicle to the contract.

EV Charging and Payment. In this phase, the charging EV meets the discharging EV to charge and pay. When they meet, they should first authenticate each other using the shared secret key. To do that as shown in Fig. 6.6, cv_i first sends a random challenge r_1, then dv_j chooses a random challenge r_2 and computes the keyed hash of the tuple $(r_1, r_2, 1)$ using the shared secret key $k_{dv_j \to cv_i}$ and sends it back to the cv_i. Then, using the shared secret key $k_{cv_i \to dv_j}$, cv_i verifies the keyed hash function sent from dv_j and then computes the keyed hash value $H_{k_{cv_i \to dv_j}}(r_1, r_2, 2)$ and

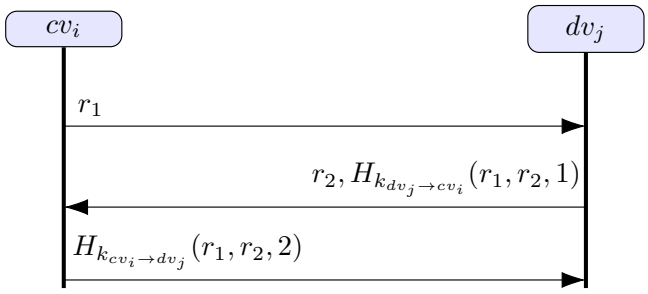


Figure 6.6: Mutual authentication between charging and discharging EVs.

sends it back to the dv_j that verifies this hash value. After charging, cv_i pays dv_i using procedure explained in Section **??**.

Coin Deposit. Finally, the last step is the *coin deposit* where the discharging EV deposits the coins it collects in a period of time (e.g., several days) in the FI. The same procedure explained in subsection 6.4.3 can be used by the discharging EV to deposit the coins.

6.5 Evaluations

In this section, we first discuss the security and privacy analysis of our schemes followed by the performance evaluations.

6.5.1 Security and Privacy Analysis

In this section, we discuss possible security and privacy concerns and how our schemes address them. Specifically, our schemes can achieve the following security/privacy objectives.

Security and transparency. Our schemes are secure against single point of failure and attack due to using blockchain (a distributed network) to run them instead of using a central server. Moreover, due to the immutability nature of the blockchain, the data stored on the blockchain cannot be tampered with by malicious adversaries. Also, our schemes ensure *transparency* of the

trading system. This is because unlike centralized systems in which it is not known how the server runs the schemes so it can favor charging stations over others to sell their offers first, in our schemes, all transactions are posted on the blockchain and the CSs and EVs can ensure that their bids/requests are received and handled properly.

Preserving the privacy of EVs' owners. In our schemes, we have used several techniques to preserve the privacy of EVs' owners such as cloaking technique, anonymous authentication scheme, one time public/private key pair and the anonymous payment. To ensure anonymity of EVs' owners, the EV owners use a random blockchain address that acts as a *pseudonym* and the generation of that address is unlinakable to the EV owner's real identity. Moreover, unlike the CS2V scheme, the private information of both the charging and discharging vehicles (location, time, and amount of energy) should be protected in the V2V scheme. This is achieved by (i) cloaking the location of the charging EV as well as the time of trading, and (ii) encrypting the submitted bids using a symmetric key encryption shared between charging and discharging EVs; therefore, the bids are *confidential* to anyone including the blockchain miners and other attackers. Also, the underlying common-prefix-linkable anonymous authentication scheme ensures that an EV can authenticate messages without being identified or linked. Finally, the anonymous payment system ensures that the coins used for the payment are untraceable and are not linked to a specific EV.

Unlinkabilty. In our schemes, no one can link a charging reservation or bid to a specific EV that sent it. This is because EVs interact with the blockchain with a randomly generated blockchain address. The blockchain address is a one-time *pseudonym* generated by the EVs, and it can not reveal the EV's real identity. Moreover, the underlying common-prefix-linkable anonymous authentication scheme ensures that an EV can authenticate messages without being linked if an EV sends one message with each prefix so the EV is anonymous and no one can link its messages. Also, in the V2V scheme, a random value a_i should change each time an EV sends a charging request so

that no one can know if the messages are sent from the same EV or not.

Anonymous and secure payment system. In our schemes, blockchain is used for storing the hash value of spent coins so that double spending can be detected. Thus, due to the use of blockchain, all spent coins are stored in tamper-resistant and non-repudiable ledger. In addition, once the coin's ownership is transferred, no one can re-spend or re-transfer the old coin because the spent coins are stored in the blockchain. Moreover, during the purchase of digital coins, the EV's real identity is used but because of using the blind signature scheme, the FI can not link the coins it signed to the EV that bought them. The payment is also secure against stolen coins. This is because if an attacker steals coins, he/she can not use them in another transactions because the spender of a digital coin has to prove with a ZKP protocol that it knows the coin's private key without revealing it.

Anonymity-yet-resistance to Sybil attacks. Our schemes ensure the anonymity of EVs owners during the energy trading because the real identity of each vehicle is not used and no one can link the messages sent from the same EV. However, if a malicious EV tries to launch a Sybil attack by pretending as multiple non-existing EVs to launch more severe attacks such as DoS on the trading system by submitting a large number of reservations/requests to the blockchain to make the system unreliable and the service is unavailable, the blockchain can detect such submissions using the linkable anonymous authentication scheme. This is because, for an EV to successfully authenticate messages, it needs to include a *prefix* to them. If an EV authenticates reservations/requests more than once for the same transaction (with the same prefix), the blockchain can link such submissions. In this way, multiple submissions can be discarded to ensure that the system is reliable and the service is available.

Mutual authentication. In our schemes, an EV, either charging or discharging, needs to authenticate messages using their secret credentials by running the *Auth* algorithm of the common

linkable anonymous authentication scheme. In addition, before starting the energy trading, a charging EV needs to prove to the CS (or the discharging EV) that it is the one that indeed made the reservation. In the CS2V, the EV and the CS engage in an instance of Schnorr's identification protocol in order for the EV to prove to the CS that it knows the private key corresponding to the public key that is computed by the EV that made the reservation. In the V2V scheme, only charging/discharging EVs which share the same symmetric secret key can successfully authenticate each other. Specifically, by verifying the keyed hash value $H_{k_{dv_j \to cv_i}}(r_1, r_2, 1)$, the cv_i can make sure that dv_j knows the shared secret key. Similarly, by verifying $H_{k_{cv_i \to dv_j}}(r_1, r_2, 2)$, dv_j can make sure that cv_i knows the shared key. This is required to ensure that no adversary can impersonate legitimate users. Moreover, the man-in-the-middle attack is impossible without the need of additional signatures. This is because any attempt from the miners to change the value g^{a_i} (g^{b_j}) to launch man-in-the-middle attack can be detected because the blockchain is transparent and EVs can ensure that the sent value g^{a_i} (and g^{b_j}) is received and stored correctly.

6.5.2 Performance Evaluation

In this section, we evaluate the performance of the proposed schemes in terms of communication and computation overheads.

Experiment Setup. We have implemented all cryptosystems used in our schemes with the C programming language, using OpenSSL's library for elliptic curve cryptography, and a 2.8 GHz Intel Core i7 CPU. For 128-bit security, we have selected ANSI's *X9.62 Prime 256v1* curve, whose order n is a 256-bit prime. We have also implemented the proposed CS2V scheme and deployed it in Ganache blockchain [187] platform, which is a private blockchain. We have also implemented the V2V scheme and deployed it in Kovan blockchain which is a testnet of the Ethereum public blockchain [2].

We have used Jsnark library[2] for building the circuits of zk-SNARKs used in the underlying common-prefix linkable anonymous authentication. The library uses libsnark [3] as a backend and has built-in gadgets, i.e., small boolean circuits including a gadget of SHA-256 and gadgets for encryption algorithms such as AES. Then, we wrote the statements of zk-SNARKs in Jsnark using the underlying gadgets. Then, the public parameters of zk-SNARK are generated using the libsnark generator library. Finally, to verify the proofs of the underlying common-prefix linkable anonymous authentication, we have used the modified Ethereum client [188] that is written in Java 1.8 with Spring framework [4].

The metrics that are used in our evaluations are the communication and computation overheads. The computation overhead is defined as the processing time required by each entity in our schemes. The communication overhead is measured by the size of messages transmitted between the entities in bytes. We also measure the gas consumption needed by our V2V scheme since the underlying blockchain is the public blockchain. *Gas* is used to quantify the execution cost associated with each transaction.

Communication and storage overhead. To measure the communication overhead, our schemes use signature scheme that uses elliptic curve cryptography (ECC). Using an elliptic curve additive group of order 256 bits, the signature's size is 64 bytes [189]. Using Table 6.1 that summarizes the size of elements used in our messages' schemes, we calculate the size of the messages sent in our schemes. Table 6.2 summarizes the total communication overhead in our schemes and the payment.

Storage overhead. For the storage overhead on the blockchain in the CS2V scheme, the blockchain stores the charging bids (150 bytes per each bid) and hashes of the spent coins (256 bytes). Thus, the total storage overhead for storing $\mathcal{M}$ bids and $\mathcal{N}$ spent coins is 150 $\times$M+256$\times$NKB.

In the V2V scheme, the blockchain stores the charging requests (741 per each request), bids

[2]https://github.com/akosba/jsnark
[3]https://github.com/scipr-lab/libsnark
[4]github.com/maxilbert/ethereumj

Table 6.1: Size of elements of our schemes.

Data	**Size (bytes)**
Timeslot	6
Location	6
Charging rate	2
Amount of energy	2
Field element	32
Group element	64
Public key	64
Signature	64
Authentication proof	729

(745 per each bid) and hash of spent coins. Thus, the total storage overhead for storing $\mathcal{L}$ requests, $\mathcal{M}$ bids and $\mathcal{N}$ spent coins is 741 $\times$L+745$\times$M+256$\times$NKB.

Communication overhead of the anonymous payment. To purchase a digital coin, the EV and the FI needs to run the blind signature scheme as discussed in Section 6.4.2. The EV sends msg_1 that contains the number and value of coins it want to purchase. Also, it sends msg_3 that has one field element. Thus, the size of msg_1 and msg_3 is 70 and 32 bytes respectively. The FI sends msg_2 that has one group point and a signature. It also need to send msg_4 that has one field element. Thus, the size of msg_1 and msg_3 is 128 and 32 bytes, respectively

Then, during the payment phase, the EV sends the transferred coin to the blockchain that contains two public keys, FI's signature, and the EV's signature. Thus, the total size of the transferred coin is 256 bytes.

To deposit the coins that the CS collects in a period of time, it needs to send 256 bytes for each transferred coin to the FI. Thus, to send n coins, the total overhead is $n \times 256$ bytes. Finally, the CS needs to prove to the FI that it owns the coin by proving that it knows the secret that corresponds to the public key in the coin using Schorr's batch protocol. To do so, the CS sends

Table 6.2: Communication overhead in our schemes.

	Entity	Message	Communication overhead "bytes"
Payment	EV	Purchasing digital coin (msg_1 and msg_3)	102
	FI	Purchasing digital coin (msg_2 and msg_4)	160
	EV	Coin transfer	256
	CS	Coin deposit	256
CS2V	CS	Submitting bid	150
	Charging EV	Reservation	795
V2V	Charging EV	Charging request	741
		Reservation	64
	Discharging EV	Charging bid	745

a commitment, i.e., a group element, then the FI sends a challenge, i.e., a field element and the CS sends a response, i.e., a field element. Thus, the total communication overhead is 128 bytes. Fig. 6.7 evaluates the effect of Schorr's batch protocol on the verifying a batch of coins instead individual ones. As described in Section 6.2, the batch protocol involves transition of the same size of messages as the original protocol, so the communication cost is constant (128 bytes) with respect to the number of coins. On the other hand, the cost of executing individual instances of Schnorr's protocol incurs a cost that grows linearly with the number of coins.

Communication overhead of the CS2V scheme. In the *submitting bids* phase, a charging station needs to send a bid that contains the following: timeslot, bid ID, location, charging rate, amount of energy, CS's public key and a signature. The total size of the packet is 150 bytes. In the *reservation* phase, a charging EV needs to send a reservation request that contains bid ID, CS's public key and authentication proof. The total size of the packet is 795 bytes.

Communication overhead of the V2V scheme. In the *charging request* phase, a charging EV needs to send a charging request that has the cloaked region and time and the authentication proof. The total size of the message is 741 bytes. In the *charging bids* phase, the discharging EV sends an offer that has ciphertext of the bid's information and the authentication proof. The total size of the

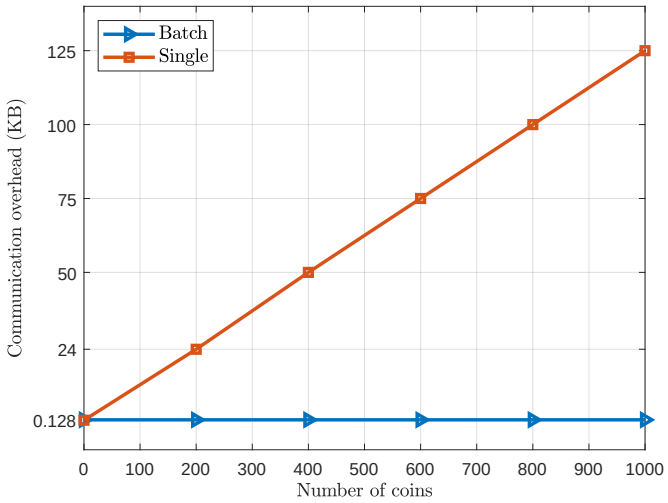


Figure 6.7: Effect of Schnorr's batch protocol on coins ownership verification overhead

Table 6.3: Computation overhead of individual operations.

Operation	**Time**
Multiplication	0.005 ms
exponentiation	4.4 ms
Signature generation	3 ms
Signature verification	1.5 ms
Encryption (AES-128)	0.0203 ms
Decryption (AES-128)	0.0078 ms
Authentication proof generation	10 s
Authentication proof verification	2.5 ms

message is 745 bytes. Finally, in the *reservation* phase, the reservation message include 64 bytes for each of public key of the selected EV.

Computation Overhead. Our measurements indicate that the multiplication and exponentiation operations take 0.005 and 4.4 ms, respectively. Note that the addition operation takes a very short time so it can be neglected. For symmetric key encryption, using AES-128, the encryption

operation takes 0.0203 ms while the decryption operation takes 0.0078 ms. For signature algorithm, the signing operation takes 3 ms while the verification takes 1.5 ms. Using Table 6.3 that summarize the computation overhead of individual operations, we calculate the computation time required to compose messages in our schemes. Table 6.4 summarizes the computation overhead of each phase in our schemes and the payment.

Anonymous payment overhead. To purchase a digital coin, an EV needs to compute two multiplication operations and one signature. Thus, the computation overhead is 3.01 ms. The FI computes two multiplication operations and one signature. Thus, the computation overhead is 3.01 ms.

To transfer a coin, the EV needs to compute a signature that takes 3 ms. Then, the validators verify two signatures, i.e., the FI's signature and the coin original owner's signature on transferred coin. Thus, the computation overhead is 6 ms. Then, to deposit a coin to the FI, the FI needs 6 ms to verify two signatures on the transferred coin. Finally, the CS needs to prove the ownership of the digital coins using Schnorr's batch protocol. In Fig. 6.8, we illustrate the efficiency of Schnorr's batch protocol in verifying the ownership of a large number of coins. It can be seen that verifying 1000 coins individually (without Schnorr's batch protocol) requires 125 ms at the prover and 250 ms at the verifier, whereas the batch protocol reduces these times to 0.1 ms and 54 ms, respectively. This indicates that Schnorr's batch protocol makes the payment process fast with less computation overhead.

CS2V scheme. In the *submitting bids* phase, the computation overhead on the charging station is 3 ms to compute a signature on the message and 1.5 ms for the validators to verify the signature. In the *reservation* phase, the charging EV computes authentication proof using the common-prefix linkable anonymous authentication scheme that needs 10 seconds. Then, the validators verify the authentication proof that needs 2.5 ms. To evaluate the computation overhead as the number of received authentication proofs increases, Table 6.5 gives the execution time of

Table 6.4: Computation overhead in our schemes.

	Entity	Message	Computation overhead
Payment	EV/FI	Purchasing digital coin	3.01 ms
	EV	Coin transfer	3 ms
	CS/FI	Coin deposit	6 ms
CS2V	Charging station	Submitting bid	3 ms
	Charging EV	Reservation	10 sec
	Blockchain	Reservation	11.31 ms
V2V	Charging EV	Charging request	10 sec
		Reservation	9 ms
	Discharging EV	Charging bid	10 sec

Table 6.5: Execution time of verifying the authentication proves.

Number of proofs	**2**	**4**	**6**	**8**	**10**
Time (ms)	10.9	15.5	16.3	17.0	17.9

verifying number of the authentication proves. It can be seen that verifying 10 proofs requires 17.9 ms and thus the on-chain computation overhead is in range of milliseconds so the on-chain computation overhead is acceptable. Although the time needed for the proof generation operation is longer than other operations, our application does not need real-time communication because EVs have enough time to compose the messages, and also the verification of the proof which is done by the blockchain needs much shorter time which is important in our application because the proof generation is done by each EV while the validators have to verify a large number of proofs sent by EVs.

V2V scheme. In the *charging request* phase, the charging vehicle computes one exponentiation operation to obtain g^{a_i} and the authentication proof. Thus, the total time is nearly 10 seconds. In the *charging bids* phase, the discharging EV needs to compute two exponentiation operations, an authentication proof, and an encryption operation to encrypt the bid. Therefore, the total time

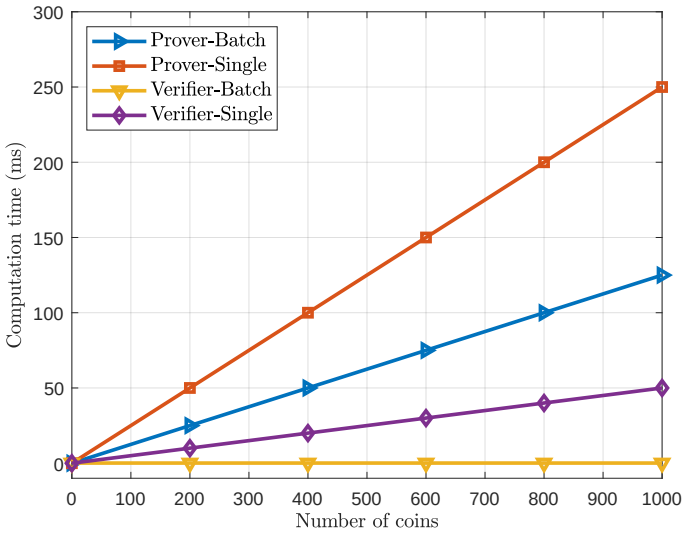


Figure 6.8: Effect of Schnorr's batch protocol on computation time of verifying the coins ownership.

Table 6.6: Execution costs of our V2V in Etherum.

Transaction	**GAS**	**Price (ETH)**
Charging request	76981	0.0003848
Charging bid	89686	0.0004834
Reservation	53981	0.0002699

is 10.04 seconds. In the *reservation* phase, the charging EV computes one exponentiation and one decryption operation to decrypt the bid. The total time is 4.4 ms

Moreover, since the V2V scheme is designed on top of public blockchain, we have also implemented the smart contract of Algorithm 6.2 in Solidity[5]. Then, it is deployed on the Kovan blockchain which is a testnet of the Ethereum blockchain [2]. In Ethereum, *gas* is used to quantify the execution cost associated with each transaction. The cost is payable using the Ethereum currency, named *Ether*. Table 6.6 summarizes the execution cost associated with each transaction in the V2V energy trading scheme. A charging vehicle needs to submit a charging request that cost 76K gas, then a discharging vehicle submits a bid that costs 89K gas, finally, 65K gas is required by

[5]https://github.com/ethereum/solidity

the charging vehicle to make the reservation. To calculate the transaction fees, according to [176], the cost for 1 unit of GAS is on average 5 Gwei and 1 Ether = 10^9 Gwei. The transaction fees associated with each transaction is given in Table 6.6. Clearly, the required fees are low for both the charging and discharging EVs.

CHAPTER 7

CONCLUSIONS AND FUTURE WORKS

In this chapter, we present our conclusions and future works.

7.1 Conclusions

In this dissertation, we have leveraged blockchain technology to design secure and privacy-preserving schemes for connected vehicles applications.

In chapter 3, a firmware update dissemination scheme leveraging blockchain and smart contract has been proposed for autonomous vehicles. A smart contract is used to ensure the authenticity and integrity of firmware updates, and more importantly to manage the reputation scores of AVs that disseminate the new updates to other AVs. The use of ABE allows AV manufacturers to target a specific set of AVs that have certain features defined by the manufacturers to download the firmware. A zero-knowledge proof protocol is used to enable the AVs to exchange an update for proof of distribution in a trust-less way. To improve the efficiency, an aggregate signature scheme is used to allow a distributor to aggregate multiple proofs to make only one transaction on the blockchain when it redeems the rewards. Our proposed scheme has been evaluated using real implementations. The experimental results indicate that the on-chain and off-chain computation overheads are in milliseconds which is enough for the moving AVs to distribute the updates successfully.

In chapter 4, we have proposed a decentralized ride sharing organization scheme. The scheme is built on top of public blockchain that provides security and transparency without the need for a trusted third party. The proposed time-locked deposit protocol ensures security against malicious behaviours of dishonest drivers/riders. In addition, the proposed reputation management system tracks drivers' behaviour, encouraging them to behave honestly. Moreover, a fair payment has

been introduced using the pay-as-you-drive methodology to discourage overcharging in the fare payment. Analysis, and experiments on Etherum blockchain were conducted to evaluate B-Ride. The results indicate that B-Ride is practical in terms of both on-chain and off-chain overheads while the momentary cost required to run the scheme is low.

In chapter 5, a charging coordination scheme for ESUs has been proposed based on the blockchain technology. First, a temporal charging coordination scheme is presented based on the Knapsack problem to maximize the power delivered to the ESUs while respecting the maximum charging capacity. Different from the traditional centralized schemes, the proposed coordination scheme is deployed on the Ethereum blockchain so that ESUs can get their charging demands in a decentralized, transparent, and verifiable manner. Finally, we found that the costs for deploying and executing the proposed smart contract in Ethereum is reasonable.

Finally, in chapter 6, we have proposed privacy-preserving schemes for CS2V and V2V energy trading. The schemes are built on top of blockchain technology that provides security and transparency without the need for a trusted third party. Moreover, launching Sybil attack by pretending as multiple non-existing EVs to launch severe attacks such DoS by submitting a large number of reservations/offers is thwarted to ensure that the energy trading system is reliable and the service is available. To preserve the privacy of EV owners, we have developed anonymous and efficient payment system using blockchain to allow EVs to pay their charging fees with untraceable digital coins. Also, the payment system is secure against stolen coins attack and fake payments. Analysis and experiments are conducted to evaluate the proposed schemes and the results indicate that our schemes are secure and can preserve the privacy of EVs' owners, and the communication and computation overheads are acceptable.

7.2 Future Work

In our future work, we will investigate the blockchain and smart contract technology in other connected vehicles applications. In vehicular networks, data contributed by vehicles can build a spatio-temporal view of traffic statistics, which can improve road safety and reduce slow traffic and jams. However, vehicles should evaluate the credibilities of messages they receive from others about traffic status and jams. As a future work, we will investigate the blockchain technology and smart contracts to develop a decentralized trust management model with incentives in vehicular networks. When there is an event, e.g., car accident, nearby vehicle will rate this event and sign it using threshold signature. Having done so, contribution of vehicles is required to provide a valid *rating score*. Then, these rating scores should be aggregated and stored in the blockchain to ensure security and transparency. Finally, to incentive vehicles to report events, a rewarding system will be investigated using smart contracts. We will also investigate the use of blockchain technology in securing unmanned aerial vehicle (UAV) that has recently gained attention from both industry and academia. UAVs can be employed for a wide range of transportation and planning applications: incident response, collect transportation surveillance data, monitor freeway conditions, emergency vehicle guidance, measurement of typical roadway usage, track vehicle movements in an intersection and monitor parking lot utilization. Using blockhchains, data collected by the UAVs can be stored on the blockchain in a tamper-resistant manner to ensure integrity and transparency.

In addition, we will explore the blockchain technology to fight COVID-19. For example, tests of COVID-19 must be carried out intelligently, and accurate data in regards to the number of tests performed needs to be maintained. To this end, blockchain technology can help in setting up distributed check-up points for testing the patients who are showing symptoms related to COVID-19. The coordinators of all these check-up points can act as nodes of the same distributed blockchain network. These nodes can continuously update data regarding the number of tests performed and

the number of laboratory-confirmed cases in their local check-up point on this network. Due to blockchain's inherent feature of being immutable, data stored in the network will be tamper-proof and can, therefore, be trusted by all the healthcare professionals.

In addition, federated learning is an emerging machine learning technique that enables distributed model training using local datasets from large-scale nodes, e.g., mobile devices, but shares only model updates without uploading the raw training data. This technique provides a promising privacy preservation for mobile devices while simultaneously ensuring high learning performance. However, relying on client-central model make the federated learning approach suffer from single point of failure and more importantly it lacks transparency since the users have no idea about how their shared updates are handled by the server. As a future work, we will explore the blockchain technology to design a secure and transparent federated learning approach. Also, an incentive mechanism that is based on the blockchain will be investigated to motivate users to participate in model learning.

Finally, cryptocurrency mining has become more popular among cyber criminals. Attackers aiming to increase their hash power by enforcing victims hardware to do mining operations so that they can increase their rewards. With a worm component, it is easier to infect hundreds of thousands or even millions of IoT devices. As a future work, we will investigate new detection algorithms to detect Crypto-mining attacks using machine learning techniques and edge computing.

REFERENCES

[1] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," *Proc. of the IEEE symposium on security and privacy (SP)*, pp. 839–858, 2016.

[2] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[3] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[4] W. Al Amiri, M. Baza, K. Banawan, M. Mahmoud, W. Alasmary, and K. Akkaya, "Towards secure smart parking system using blockchain technology," *Proc. of 17th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las vegas, USA*, 2020.

[5] Hyperledger. [Online]. Available: https://www.hyperledger.org.

[6] F. P. Junqueira, B. C. Reed, and M. Serafini, "Zab: High-performance broadcast for primary-backup systems," *in Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Hong Kong, China*, pp. 245–256, 2011.

[7] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," *in Proceedings of the USENIX Annual Technical Conference (USENIX ATC), Philadelphia, PA, USA*, pp. 305–319, 2014.

[8] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," *in Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA*, pp. 173–186, 1999.

[9] M. Pazos-Revilla, A. Alsharif, S. Gunukula, T. N. Guo, M. Mahmoud, and X. Shen, "Secure and privacy-preserving physical-layer-assisted scheme for EV dynamic charging system," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 4, pp. 3304–3318, 2018.

[10] M. Baza, j. Baxter, N. Lasla, M. Mahmoud, M. Abdallah, and M. Younis, "Incentivized and secure blockchain-based firmware update and dissemination for autonomous vehicles," *Transportation and Power Grid in Smart Cities: Communication Networks and Services*, 2020.

[11] A. Sherif, A. Alsharif, J. Moran, and M. Mahmoud, "Privacy-preserving ride sharing organization scheme for autonomous vehicles in large cities," *Proceedings of the IEEE 86th Vehicular Technology Conference, IEEE VTC2017-Fall*, September 2017.

[12] T. Lassa. The beginning of the end of driving. [Online]. Available: http://www.motortrend.com/news/the-beginning-of-theend-of-driving/

[13] J. Eriksson, H. Balakrishnan, and S. Madden, "Cabernet: vehicular content delivery using wifi," *Proceedings of the 14th ACM international conference on Mobile computing and networking*, pp. 199–210, 2008.

[14] M. Baza, M. Nabil, N. Bewermeier, K. Fidan, M. Mahmoud, and M. Abdallah, "Detecting sybil attacks using proofs of work and location in VANETs," *IEEE Transactions on Dependable and Secure Computing*, In press, In press, May 2020.

[15] I. Vakilinia, S. Badsha, and S. Sengupta, "Crowdfunding the insurance of a cyber-product using blockchain," pp. 964–970, 2018.

[16] I. Vakilinia, S. Badsha, E. Arslan, and S. Sengupta, "Pooling approach for task allocation in the blockchain based decentralized storage network," 2019.

[17] M. Baza, N. Lasla, M. Mahmoud, G. Srivastava, and M. Abdallah, "B-Ride: Ride sharing with privacy-preservation, trust and fair payment atop public blockchain," *IEEE Transactions on Network Science and Engineering*, In press, Dec. 2019.

[18] A. greeenberg. [Online]. Available: https://www.wired.com/2015/07/jeep-hack-chrysler-recalls-1-4m-vehicles-bug-fix/

[19] G. Lin, D. Fu, J. Zhu, and G. Dasmalchi, "Cloud computing: It as a service," *IT professional*, no. 2, pp. 10–13, 2009.

[20] U.s. autonomous car market. [Online]. Available: https://www.businesswire.com/news/home/20180831005410/en/U.S.-Autonomous-Car-Market-2018-2023---20.8

[21] R. Amer, A. A. El-Sherif, H. Ebrahim, and A. Mokhtar, "Stability analysis for multi-user cooperative cognitive radio network with energy harvesting," in *IEEE International Conference on Computer and Communications (ICCC)*, Oct 2016, pp. 2369–2375.

[22] R. Amer, A. A. El-sherif, H. Ebrahim, and A. Mokhtar, "Cooperation and underlay mode selection in cognitive radio network," in *International Conference on Future Generation Communication Technologies (FGCT)*, Aug 2016, pp. 36–41.

[23] R. Amer, A. A. El-Sherif, H. Ebrahim, and A. Mokhtar, "Cooperative cognitive radio network with energy harvesting: Stability analysis," in *International Conference on Computing, Networking and Communications (ICNC)*, Feb 2016, pp. 1–7.

[24] B. Galkin, R. Amer, E. Fonseca, and L. A. DaSilva, "Intelligent uav base station selection in urban environments: A supervised learning approach," *arXiv preprint arXiv:2003.01287*, 2020.

[25] R. Amer, W. Saad, and N. Marchetti, "Mobility in the sky: Performance and mobility analysis for cellular-connected uavs," *IEEE Transactions on Communications*, pp. 1–1, 2020.

[26] R. Amer, W. Saad, B. Galkin, and N. Marchetti, "Performance analysis of mobile cellular-connected drones under practical antenna configurations," in *Proc. of IEEE International Conference on Communications (ICC)*, Dublin, June. 2020.

[27] R. Amer, H. Elsawy, M. M. Butt, E. A. Jorswieck, M. Bennis, and N. Marchetti, "Optimized caching and spectrum partitioning for d2d enabled cellular systems with clustered devices," *IEEE Transactions on Communications*, pp. 1–1, 2020.

[28] R. Amer, H. Elsawy, J. Kibiłda, M. M. Butt, and N. Marchetti, "Performance analysis and optimization of cache-assisted CoMP for clustered D2D networks," *submitted to IEEE Transactions on Mobile Computing*, 2019.

[29] R. Amer, H. Elsawy, M. M. Butt, E. A. Jorswieck, M. Bennis, and N. Marchetti, "Optimizing joint probabilistic caching and communication for clustered D2D networks," *arXiv preprint arXiv:1810.05510*, 2018.

[30] C. Chaccour, R. Amer, B. Zhou, and W. Saad, "On the reliability of wireless virtual reality at terahertz (THz) frequencies," in *10th IFIP International Conference on New Technologies*, Spain, June. 2019.

[31] R. Amer, M. M. Butt, M. Bennis, and N. Marchetti, "Inter-cluster cooperation for wireless D2D caching networks," *IEEE Transactions on Wireless Communications*, vol. 17, no. 9, pp. 6108–6121, Sep. 2018.

[32] R. Amer, M. M. Butt, H. ElSawy, M. Bennis, J. Kibilda, and N. Marchetti, "On minimizing energy consumption for D2D clustered caching networks," in *IEEE Global Communications Conference (GLOBECOM)*, Dec 2018, pp. 1–6.

[33] R. Amer, W. Saad, H. ElSawy, M. Butt, and N. Marchetti, "Caching to the sky: Performance analysis of cache-assisted CoMP for cellular-connected UAVs," in *Proc. of the IEEE Wireless Communications and Networking Conference (WCNC)*, Marrakech, Morocco, April. 2019.

[34] R. Amer, H. ElSawy, J. Kibiłda, M. M. Butt, and N. Marchetti, "Cooperative transmission and probabilistic caching for clustered D2D networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*, April 2019, pp. 1–6.

[35] R. Amer, M. M. Butt, and N. Marchetti, "Optimizing joint probabilistic caching and channel access for clustered {D2D} networks," *arXiv preprint arXiv:2003.02676*, 2020.

[36] A. Sherif, A. Alsharif, J. Moran, and M. Mahmoud, "Privacy-preserving autonomous cab service management scheme," *Proceedings of the 3rd Africa and Middle East Conference on Software Engineering*, December 2017.

[37] Z. Haddad, A. Alsharif, A. Sherif, and M. Mahmoud, "Privacy-preserving intra-MME group handover via MRN in LTE-A networks for repeated trips," *Proceedings of the IEEE 86th Vehicular Technology Conference (VTC-Fall)*, 2017.

[38] D. Schrank, B. Eisele, and T. Lomax, "2014 urban mobility report: powered by Inrix traffic data," Tech. Rep., 2015.

[39] D. Sánchez, S. Martínez, and J. Domingo-Ferrer, "Co-utile P2P ridesharing via decentralization and reputation management," *Transportation Research Part C: Emerging Technologies*, vol. 73, pp. 147–166, June 2016.

[40] Ride sharing market cap. [Online]. Available: https://www.globenewswire.com/news-release/2019/01/17/1701096/0/en/218-Billion-Ride-Sharing-Market-Global-Forecast-to-2025.html

[41] M. Baza, M. Nabil, M. Ismail, M. Mahmoud, E. Serpedin, and M. Rahman, "Blockchain-based charging coordination mechanism for smart grid energy storage units," *Proc. of IEEE International Conference on Blockchain, Atlanta, USA*, 2019.

[42] A. Alsharif, S. Tonyali, M. Mahmoud, K. Akkaya, M. I. Muhammad, and E. Serpedin, "Performance analysis of certificate renewal scheme for AMI networks," *Proceedings of the 7th International Workshop on Computer Science and Engineering, WCSE 2017*, pp. 968–976, 2017.

[43] Uber data breach. [Online]. Available: https://www.npr.org/2018/09/27/652119109/uber-pays-148-million-over-year-long-cover-up-of-data-breach

[44] Motherboard: Uber china statement on service outage. [Online]. Available: https://motherboard.vice.com/en_us/article/3daa55/ubers-china-problem

[45] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J.-N. Liu, Y. Xiang, and R. Deng, "Crowdbc: A blockchain-based decentralized framework for crowdsourcing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 6, pp. 1251–1266, June 2019.

[46] S. Badsha, I. Vakilinia, and S. Sengupta, "Blocynfo-share: Blockchain based cybersecurity information sharing with fine grained access control," 2020.

[47] V. B. Venkateswaran, D. K. Saini, and M. Sharma, "Approaches for optimal planning of the energy storage units in distribution network and their impacts on system resiliency—a review," *CSEE Journal of Power and Energy Systems*, In press, 2020.

[48] M. Wang, M. Ismail, R. Zhang, X. Shen, E. Serpedin, and K. Qaraqe, "Spatio-temporal coordinated V2V energy swapping strategy for mobile PEVs," *IEEE Transactions on Smart Grid*, vol. 9, no. 3, pp. 1566–1579, 2018.

[49] M. M. Mahmoud, X. Lin, and X. Shen, "Secure and reliable routing protocols for heterogeneous multihop wireless networks," *IEEE transactions on parallel and distributed systems*, vol. 26, no. 4, pp. 1140–1153, 2013.

[50] M. E. Mahmoud and X. Shen, "ESIP: Secure incentive protocol with limited use of public-key cryptography for multihop wireless networks," *IEEE Transactions on Mobile Computing*, vol. 10, no. 7, pp. 997–1010, 2010.

[51] E. Sortomme, M. M. Hindi, S. J. MacPherson, and S. Venkata, "Coordinated charging of plug-in hybrid electric vehicles to minimize distribution system losses," *IEEE transactions on smart grid*, vol. 2, no. 1, pp. 198–205, 2011.

[52] R. A. Verzijlbergh, M. Grond, Z. Lukszo, J. G. Slootweg, and M. D. Ilic, "Network impacts and cost savings of controlled EV charging," *IEEE Transactions on Smart Grid*, vol. 3, no. 3, pp. 1203–1212, 2012.

[53] M. Wang, M. Ismail, R. Zhang, X. S. Shen, E. Serpedin, and K. Qaraqe, "A semi-distributed V2V fast charging strategy based on price control," *Proc. of IEEE Global Communications Conference (GLOBECOM)*, pp. 4550–4555, 2014.

[54] P. Akula, M. Mahmoud, K. Akkaya, and M. Song, "Privacy-preserving and secure communication scheme for power injection in smart grid," *Proc. of IEEE International Conference on Smart Grid Communications*, 2015.

[55] NIST, "Guidelines for smart grid cyber security: vol. 3 supportive analyses and references," *NISTIR 7628, The Smart Grid Interoperability Panel - Cyber Security Working Group*, August 2010.

[56] A. Jakaria, W. Yang, B. Rashidi, C. Fung, and M. A. Rahman, "Vfence: A defense against distributed denial of service attacks using network function virtualization," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. IEEE, 2016, pp. 431–436.

[57] A. Jakaria, B. Rashidi, M. A. Rahman, C. Fung, and W. Yang, "Dynamic ddos defense resource allocation using network function virtualization," in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, 2017, pp. 37–42.

[58] A. Jakaria and M. A. Rahman, "A formal framework of resource management for vnfaas in cloud," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE, 2017, pp. 254–261.

[59] A. Jakaria, M. A. Rahman, and C. Fung, "A requirement-oriented design of nfv topology by formal synthesis," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1739–1753, 2019.

[60] A. Jakaria, M. A. Rahman, and C. J. Fungt, "Automated synthesis of nfv topology: A security requirement-oriented design," in *2017 13th International Conference on Network and Service Management (CNSM)*. IEEE, 2017, pp. 1–5.

[61] A. Jakaria, M. A. Rahman, and A. Gokhale, "A formal model for resiliency-aware deployment of sdn: A scada-based case study," in *2019 15th International Conference on Network and Service Management (CNSM)*. IEEE, 2019, pp. 1–5.

[62] J. Wang, G. R. Bharati, S. Paudyal, O. Ceylan, B. P. Bhattarai, and K. S. Myers, "Coordinated electric vehicle charging with reactive power support to distribution grids," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 1, pp. 54–63, Jan 2019.

[63] K. Rabieh, M. M. Mahmoud, K. Akkaya, and S. Tonyali, "Scalable certificate revocation schemes for smart grid Ami networks using bloom filters," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 4, pp. 420–432, 2015.

[64] S. Tonyali, O. Cakmak, K. Akkaya, M. M. Mahmoud, and I. Guvenc, "Secure data obfuscation scheme to enable privacy-preserving state estimation in smart grid AMI networks," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 709–719, 2015.

[65] A. Beussink, K. Akkaya, I. F. Senturk, and M. Mahmoud, "Preserving consumer privacy on IEEE 802.11 s-based smart grid AMI networks using data obfuscation," *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 658–663, 2014.

[66] M. Mahmoud, J. Misic, and X. Shen, "Efficient public-key certificate revocation schemes for smart grid," *Proc. of IEEE Global Communications Conference (GLOBECOM)*, pp. 778–783, 2013.

[67] D. T. Hoang, P. Wang, D. Niyato, and E. Hossain, "Charging and discharging of plug-in electric vehicles (PEVs) in vehicle-to-grid (V2G) systems: A cyber insurance-based model," *IEEE Access*, vol. 5, pp. 732–754, 2017.

[68] R. Alvaro-Hermana, J. Fraile-Ardanuy, P. J. Zufiria, L. Knapen, and D. Janssens, "Peer to peer energy trading with electric vehicles," *IEEE Intelligent Transportation Systems Magazine*, vol. 8, no. 3, pp. 33–44, 2016.

[69] F. Knirsch, A. Unterweger, and D. Engel, "Privacy-preserving blockchain-based electric vehicle charging with dynamic tariff decisions," *Computer Science - R&D*, vol. 33, no. 1-2, pp. 71–79, 2018.

[70] J. Kang, R. Yu, X. Huang, S. Maharjan, Y. Zhang, and E. Hossain, "Enabling localized peer-to-peer electricity trading among plug-in hybrid electric vehicles using consortium blockchains," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 6, pp. 3154–3164, 2017.

[71] C. Zhang, L. Zhu, C. Xu, C. Zhang, K. Sharif, H. Wu, and H. Westermann, "BSFP: Blockchain-enabled smart parking with fairness, reliability and privacy protection," *IEEE Transactions on Vehicular Technology*, Early access, 2020.

[72] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," *Proc. of IEEE Symposium on Security and Privacy (SP)*, pp. 459–474, 2014.

[73] Countries that have banned cryptocurrency. [Online]. Available: https://www.escapeartist.com/blog/countries-banned-bitcoin/

[74] M. Pustišek, A. Kos, and U. Sedlar, "Blockchain based autonomous selection of electric vehicle charging station," *Proc. of international conference on identification, information and knowledge in the Internet of Things (IIKI)*, pp. 217–222, 2016.

[75] Z. Su, Y. Wang, Q. Xu, M. Fei, Y.-C. Tian, and N. Zhang, "A secure charging scheme for electric vehicles with smart communities in energy blockchain," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4601–4613, 2018.

[76] X. Huang, Y. Zhang, D. Li, and L. Han, "An optimal scheduling algorithm for hybrid EV charging scenario using consortium blockchains," *Future Generation Computer Systems*, vol. 91, pp. 555–562, 2019.

[77] E. O. Augustine, O. E. Stephen, and I. S. Ityokumbul, "Design fabrication and testing of a viscometer for testing viscosity of liquids," *International Journal of Engineering Research & Technology*, vol. 8, no. 5, pp. 659–663, 2019.

[78] B. Sunday, U. Thomas, I. Ityokumbul, and A. Nasir, "Effect of number of passengers [loading] on centre of gravity of a three wheeled vehicle [keke-napep]," *ARID ZONE JOURNAL OF ENGINEERING, TECHNOLOGY AND ENVIRONMENT*, vol. 15, no. 3, pp. 638–647, 2019.

[79] A. N. Mohammed, Z. Y. Ladan, and S. I. Igbax, "Analysis of the design parameters of a cyclone dust separator," *Journal of Engineering and Applied Scientific Research*, vol. 11, no. 1, pp. 34–46, 2019.

[80] I. S. Ityokumbul, N. N. Yunusa, and O. E. Augustine, "The use of cooking gas as refrigerant in a domestic refrigerator," *Int. Journal of Engineering Research and Applications*, vol. 6, no. 3, pp. 19–25, 2016.

[81] S. I. Igbax, E. A. Ogwu, and L. I. Atiku, "Causes of failure in storage facilities and their supports," *The International Journal Of Engineering And Science (IJES)*, vol. 5, no. 5, pp. 14–18, 2016.

[82] H. Mohammed, T. A. Odetola, and S. R. Hasan, "How secure is distributed convolutional neural network on iot edge devices?" *arXiv preprint arXiv:2006.09276*, 2020.

[83] H. Mohammed, T. A. Odetola, S. R. Hasan, S. Stissi, I. Garlin, and F. Awwad, "(hiadiot): Hardware intrinsic attack detection in internet of things; leveraging power profiling," in *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2019, pp. 852–855.

[84] T. A. Odetola, K. M. Groves, and S. R. Hasan, "2l-3w: 2-level 3-way hardware-software co-verification for the mapping of deep learning architecture (dla) onto fpga boards," *arXiv preprint arXiv:1911.05944*, 2019.

[85] T. A. Odetola, O. Oderhohwo, and S. R. Hasan, "A scalable multilabel classification to deploy deep learning architectures for edge devices," *arXiv preprint arXiv:1911.02098*, 2019.

[86] T. A. Odetola, H. R. Mohammed, and S. R. Hasan, "A stealthy hardware trojan exploiting the architectural vulnerability of deep learning architectures: Input interception attack (iia)," *arXiv preprint arXiv:1911.00783*, 2019.

[87] G. Lee, H. Kim, Y. Cho, and S.-H. Lee, "Qoe-aware scheduling for sigmoid optimization in wireless networks," *IEEE Communications Letters*, vol. 18, no. 11, pp. 1995–1998, 2014.

[88] G. Lee, W. Saad, M. Bennis, A. Mehbodniya, and F. Adachi, "Online ski rental for on/off scheduling of energy harvesting base stations," *IEEE Transactions on Wireless Communications*, vol. 16, no. 5, pp. 2976–2990, 2017.

[89] ——, "Online ski rental for scheduling self-powered, energy harvesting small base stations," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.

[90] G. Lee, H. Kim, Y.-T. Kim, and B.-H. Kim, "Delaunay triangulation based green base station operation for self organizing network," in *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. IEEE, 2013, pp. 1–6.

[91] M. I. Baza, M. M. Fouda, A. S. T. Eldien, and H. A. Mansour, "An efficient distributed approach for key management in microgrids," *Proc. of the Computer Engineering Conference (ICENCO)*, pp. 19–24, 2015.

[92] M. Baza, M. Pazos-Revilla, M. Nabil, A. Sherif, M. Mahmoud, and W. Alasmary, "Privacy-preserving and collusion-resistant charging coordination schemes for smart grid," *arXiv preprint arXiv:1905.04666*, 2019.

[93] A. Shafee, M. Baza, D. A. Talbert, M. M. Fouda, M. Nabil, and M. Mahmoud, "Mimic learning to generate a shareable network intrusion detection model," *Proc. of 17th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las vegas, USA*, 2020.

[94] M. Baza, M. Fouda, M. Nabil, A. S. Tag, H. Mansour, and M. Mahmoud, "Blockchain-based distributed key management approach tailored for smart grid," 2019.

[95] D. Kim, H. Nam, and D. Kim, "Adaptive code dissemination based on link quality in wireless sensor networks," *IEEE Internet of Things Journal*, vol. 4, no. 3, pp. 685–695, 2017.

[96] P. K. Dutta, J. W. Hui, D. C. Chu, and D. E. Culler, "Securing the deluge network programming system," *Proc. of ACM international conference on Information processing in sensor networks*, pp. 326–333, 2006.

[97] B. Lee and J.-H. Lee, "Blockchain-based secure firmware update for embedded devices in an internet of things environment," *The Journal of Supercomputing*, vol. 73, no. 3, pp. 1152–1167, 2017.

[98] O. Leiba, Y. Yitzchak, R. Bitton, A. Nadler, and A. Shabtai, "Incentivized delivery network of iot software updates based on trustless proof-of-distribution," *Proc. of IEEE European Symposium on Security and Privacy Workshops*, pp. 29–39, 2018.

[99] D. K. Nilsson and U. E. Larson, "Secure firmware updates over the air in intelligent vehicles," *Proc. of the IEEE International Conference on communications on*, pp. 380–384, 2008.

[100] A. Boudguiga, N. Bouzerna, L. Granboulan, A. Olivereau, F. Quesnel, A. Roger, and R. Sirdey, "Towards better availability and accountability for IoT updates by means of a blockchain," *Security and Privacy Workshops (EuroS&PW), 2017 IEEE European Symposium on*, pp. 50–58, 2017.

[101] W. Al Amiri, M. Baza, M. Mahmoud, K. Banawan, W. Alasmary, and K. Akkaya, "Privacy-preserving smart parking system using blockchain and private information retrieval," *Proc. of the IEEE International Conference on Smart Applications, Communications and Networking (SmartNets 2019)*, 2020.

[102] I. Yilmaz and R. Masum, "Expansion of cyber attack data from unbalanced datasets using generative techniques," *arXiv preprint arXiv:1912.04549*, 2019.

[103] I. Yilmaz, "Practical fast gradient sign attack against mammographic image classifier," *arXiv preprint arXiv:2001.09610*, 2020.

[104] I. Yilmaz, R. Masum, and A. Siraj, "Addressing imbalanced data problem with generative adversarial network for intrusion detection," 2020.

[105] I. Yilmaz, A. Siraj, and D. Ulybyshev, "Improving dga-based malicious domain classifiers for malware defense with adversarial machine learning," 2020.

[106] I. Yilmaz and A. Siraj, "Avoiding occupancy detection from smart meter using adversarial machine learning," *Information*, 2020.

[107] M. Baza, M. Mahmoud, G. Srivastava, W. Alasmary, and M. Younis, "A light blockchain-powered privacy-preserving organization scheme for ride sharing services," *Proc. of the IEEE 91th Vehicular Technology Conference (VTC-Spring), Antwerp, Belgium*, May 2020.

[108] M. Asghari, D. Deng, C. Shahabi, U. Demiryurek, and Y. Li, "Price-aware real-time ride-sharing at scale: an auction-based approach," *Proc. of ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 33–40, 2016.

[109] A. Pham, I. Dacosta, M. Endignoux, and J. Hubaux, "Oride: A privacy-preserving yet accountable ride-hailing service," *Proc. of the 26th USENIX Security Symposium, BC, CANADA*, 2017.

[110] M. Rigby, A. Krüger, and S. Winter, "An opportunistic client user interface to support centralized ride share planning," *Proc. of the 21st ACM SIGSPATIAL international conference on advances in geographic information systems, Florida, USA*, pp. 34–43, 2013.

[111] Dacsee platform. [Online]. Available: https://dacsee.com/

[112] Arcade city. [Online]. Available: https://arcade.city/

[113] U. M. Aïvodji, K. Huguenin, M.-J. Huguet, and M.-O. Killijian, "Sride: A privacy-preserving ridesharing system," *Proc. of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pp. 40–50, 2018.

[114] A. Ferdowsi, A. Eldosouky, and W. Saad, "Interdependence-aware game-theoretic framework for secure intelligent transportation systems," *arXiv preprint arXiv:2007.05904*, 2020.

[115] A. French, M. Mozaffari, A. Eldosouky, and W. Saad, "Environment-aware deployment of wireless drones base stations with google earth simulator," *arXiv preprint arXiv:1805.10424*, 2018.

[116] A. Eldosouky, W. Saad, and N. Mandayam, "Resilient critical infrastructure: Bayesian network analysis and contract-based optimization," *arXiv preprint arXiv:1709.00303*, 2017.

[117] A. Eldosouky, W. Saad, C. Kamhoua, and K. Kwiat, "Contract-theoretic resource allocation for critical infrastructure protection," in *2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–6.

[118] A. Eldosouky and W. Saad, "On the cybersecurity of m-health iot systems with led bitslice implementation," in *2018 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 2018, pp. 1–6.

[119] A. Eldosouky, W. Saad, and D. Niyato, "Single controller stochastic games for optimized moving target defense," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.

[120] A. Eldosouky, A. Ferdowsi, and W. Saad, "Drones in distress: A game-theoretic countermeasure for protecting uavs against gps spoofing," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 2840–2854, 2019.

[121] J. Ni, K. Zhang, X. Lin, H. Yang, and X. S. Shen, "AMA: Anonymous mutual authentication with traceability in carpooling systems," *Proc. of the 2016 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2016.

[122] M. H. Au, W. Susilo, and Y. Mu, "Constant-size dynamic k-TAA," *Proc. of the International Conference on Security and Cryptography for Networks*, pp. 111–125, 2006.

[123] A. B. Sherif, K. Rabieh, M. Mahmoud, and X. Liang, "Privacy-preserving ride sharing scheme for autonomous vehicles in big data era," *IEEE Internet of Things Journal*, vol. 4, no. 2, pp. 611–618, 2016.

[124] S. Kudva, R. Norderhaug, S. Badsha, S. Sengupta, and A. Kayes, "Pebers: Practical ethereum blockchain based efficient ride hailing service," in *IEEE International Conference on Informatics, IoT and Enabling Technologies*, 2020.

[125] S. Maskey, S. Badsha, S. Sengupta, and I. Khalil, "Bits: Blockchain based intelligent transportation system with outlier detection for smart city," in *2020 IEEE 18th Annual International Conference on Pervasive Computing and Communication Worshop*, 2020.

[126] A. Bhattacharjee, S. Badsha, A. Shahid, H. Livani, and S. Sengupta, "Block-phasor: A decentralized blockchain framework to enhance security of synchrophasor," in *IEEE Kansas Power and Energy Conference, Manhattan, Kansas, USA*, 2020.

[127] Y. Yuan and F.-Y. Wang, "Towards blockchain-based intelligent transportation systems," *in Proc. of the IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, Brazil*, pp. 2663–2668, Nov. 2016.

[128] Y. Semenko and D. Saucez, "Distributed privacy preserving platform for ridesharing services," Ph.D. dissertation, Inria-Sophia Antipolis university, 2019.

[129] M. Li, L. Zhu, and X. Lin, "Efficient and privacy-preserving carpooling using blockchain-assisted vehicular fog computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4573–4584, June 2019.

[130] G. T. Reddy, M. P. K. Reddy, K. Lakshmanna, D. S. Rajput, R. Kaluri, and G. Srivastava, "Hybrid genetic algorithm and a fuzzy logic classifier for heart disease diagnosis," *Evolutionary Intelligence*, vol. 13, no. 2, pp. 185–196, 2020.

[131] J. Sakhnini, H. Karimipour, A. Dehghantanha, R. M. Parizi, and G. Srivastava, "Security aspects of internet of things aided smart grids: A bibliometric survey," *Internet of things*, p. 100111, 2019.

[132] G. T. Reddy, M. P. K. Reddy, K. Lakshmanna, R. Kaluri, D. S. Rajput, G. Srivastava, and T. Baker, "Analysis of dimensionality reduction techniques on big data," *IEEE Access*, vol. 8, pp. 54 776–54 788, 2020.

[133] L. Malina, G. Srivastava, P. Dzurenda, J. Hajny, and R. Fujdiak, "A secure publish/subscribe protocol for internet of things," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, 2019, pp. 1–10.

[134] Y. Ota, H. Taniguchi, T. Nakajima, K. M. Liyanage, J. Baba, and A. Yokoyama, "Autonomous distributed V2G (vehicle-to-grid) satisfying scheduled charging," *IEEE Transactions on Smart Grid*, vol. 3, no. 1, pp. 559–564, 2012.

[135] W. Tushar, J. Zhang, D. Smith, H. Poor, and S. Thiebaux, "Prioritizing consumers in smart grid: A game theoretic approach," *IEEE Transactions on Smart Grid*, vol. 5, no. 3, pp. 1429–1438, May 2014.

[136] E. Sortomme and M. El-Sharkawi, "Optimal scheduling of vehicle-to-grid energy and ancillary services," *IEEE Transactions on Smart Grid*, vol. 3, no. 1, pp. 351–359, March 2012.

[137] I. H. Sarker, A. Kayes, S. Badsha, H. Alqahtani, P. Watters, and A. Ng, "Cybersecurity data science: an overview from machine learning perspective," *Journal of Big Data*, vol. 7, no. 1, pp. 1–29, 2020.

[138] S. Badsha, X. Yi, I. Khalil, D. Liu, S. Nepal, and K.-Y. Lam, "Privacy preserving user based web service recommendations," *IEEE Access*, vol. 6, pp. 56 647–56 657, 2018.

[139] S. Badsha, I. Khalil, X. Yi, and M. Atiquzzaman, "Designing privacy-preserving protocols for content sharing and aggregation in content centric networking," *IEEE Access*, vol. 6, pp. 42 119–42 130, 2018.

[140] S. Badsha, I. Vakilinia, and S. Sengupta, "Privacy preserving cyber threat information sharing and learning for cyber defense," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2019, pp. 0708–0714.

[141] S. Badsha, X. Yi, and I. Khalil, "A practical privacy-preserving recommender system," *Data Science and Engineering*, vol. 1, no. 3, pp. 161–177, 2016.

[142] A. Thakkar, S. Badsha, and S. Sengupta, "Game theoretic approach applied in cybersecurity information exchange framework," in *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*, 2020.

[143] S. Badsha, X. Yi, I. Khalil, and E. Bertino, "Privacy preserving user-based recommender system," in *2017 IEEE 37th international conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 1074–1083.

[144] S. Badsha, X. Yi, I. Khalil, D. Liu, S. Nepal, E. Bertino, and K.-Y. Lam, "Privacy preserving location-aware personalized web service recommendations," *IEEE Transactions on Services Computing*, 2018.

[145] M. Mahmoud, M. Ismail, P. Akula, K. Akkaya, E. Serpedin, and K. Qaraqe, "Privacy-aware power charging coordination in future smart grid," *Proc. of the Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, 2016.

[146] N. Z. Aitzhan and D. Svetinovic, "Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 5, pp. 840–852, 2018.

[147] S. Gunukula, A. B. T. Sherif, M. Pazos-Revilla, B. Ausby, M. Mahmoud, and X. S. Shen, "Efficient scheme for secure and privacy-preserving electric vehicle dynamic charging system," *Proc. IEEE International Conference on Communications (ICC)*, pp. 1–6, 2017.

[148] L. Zhu, M. Li, Z. Zhang, and Z. Qin, "ASAP: An anonymous smart-parking and payment scheme in vehicular networks," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, Early access, 2018.

[149] M. H. Au, J. K. Liu, J. Fang, Z. L. Jiang, W. Susilo, and J. Zhou, "A new payment system for enhancing location privacy of electric vehicles," *IEEE Transactions on Vehicular Technology*, vol. 63, no. 1, pp. 3–18, 2014.

[150] Y. Rouselakis and B. Waters, "Efficient statically-secure large-universe multi-authority attribute-based encryption," *Proc. of the International Conference on Financial Cryptography and Data Security*, pp. 315–332, 2015.

[151] Y. Lu, Q. Tang, and G. Wang, "Zebralancer: Private and anonymous crowdsourcing system atop open blockchain," *Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS)*, pp. 853–865, 2018.

[152] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture." *USENIX Security Symposium*, pp. 781–796, 2014.

[153] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," *Proc. of springer International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 416–432, 2003.

[154] M. Baza and M. Mahmoud, "Privacy-preserving blockchain-based energy trading schemes for electric vehicles," *IEEE Access*, 2020.

[155] M. Baza, A. Salazar, M. Mahmoud, M. Abdallah, and K. Akkaya, "On sharing models instead of data using mimic learning for smart health applications," *Proc. of the IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*, pp. 231–236, 2020.

[156] M. Baza, M. Ismail, M. Mahmoud, E. Serpedin, and M. Rahman, "Blockchain-based privacy-preserving charging coordination mechanism for energy storage units," *arXiv preprint arXiv:1811.02001*, 2019.

[157] Precomiled contract for BGLS signature, "https://github.com/project-arda/bgls-on-evm."

[158] Ethereum PoA private network. [Online]. Available: https://github.com/a1brz/eth-private-network

[159] Node.js javascript runtime. [Online]. Available: https://nodejs.org/en/

[160] M. Baza, M. Nabil, N. Lasla, K. Fidan, M. Mahmoud, and M. Abdallah, "Blockchain-based firmware update scheme tailored for autonomous vehicles," *Proc. of the IEEE Wireless Communications and Networking Conference (WCNC), Marrakech, Morocco*, April 2019.

[161] M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity," *Proc. of Springer International Conference on the Theory and Application of Cryptology and Information Security*, pp. 191–219, 2016.

[162] J. Camenisch, R. Chaabouni, and a. shelat, "Efficient protocols for set membership and range proofs," *in Proc. of the International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, VIC, Australia*, pp. 234–252, 2008.

[163] J. Wu, P. Ma, and K. L. Xie, "In sharing economy we trust: the effects of host attributes on short-term rental purchases," *International Journal of Contemporary Hospitality Management*, vol. 29, no. 11, pp. 2962–2976, Nov. 2017.

[164] F. Boeira, M. Asplund, and M. P. Barcellos, "Vouch: A secure proof-of-location scheme for VANETs," *Proc. of the ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 241–248, 2018.

[165] M. Furuhata, M. Dessouky, F. Ordóñez, M.-E. Brunet, X. Wang, and S. Koenig, "Ridesharing: The state-of-the-art and future directions," *Transportation Research Part B: Methodological*, vol. 57, pp. 28–46, 2013.

[166] Uber overcharging. [Online]. Available: https://www.wspa.com/news/uber-driver-off-the-job-after-he-charged-for-fake-puke-2/1018463150

[167] Binomial coefficient. [Online]. Available: https://en.wikipedia.org/wiki/Binomial_coefficient

[168] F. Knirsch, A. Unterweger, and D. Engel, "Privacy-preserving blockchain-based electric vehicle charging with dynamic tariff decisions," *Computer Science-Research and Development*, vol. 33, no. 1-2, pp. 71–79, 2018.

[169] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," *in Proc. of the International Cryptology Conference, California, USA*, pp. 421–439, 2014.

[170] D. Boneh and X. Boyen, "Short signatures without random oracles," *Proc. of International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland*, pp. 56–73, 2004.

[171] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," *Proc. of Springer Annual International Cryptology Conference*, pp. 129–140, 1991.

[172] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," *Proc. of Conference on the Theory and Application of Cryptographic Techniques*, pp. 186–194, 1986.

[173] Kovan etherum test net. [Online]. Available: https://kovan.etherscan.io

[174] H. S. Galal and A. M. Youssef, "Succinctly verifiable sealed-bid auction smart contract," *Proc. of 13th International Workshop on Data Privacy Management, Barcelona, Spain*, pp. 3–19, 2018.

[175] H. Duan, Y. Zheng, Y. Du, A. Zhou, C. Wang, and M. H. Au, "Aggregating crowd wisdom via blockchain: A private, correct, and robust realization," *Proc. of IEEE International Conference on Pervasive Computing and Communications (PerCom2019)*, pp. 43–52, 2019.

[176] Etherum gas station, "Available: https://ethgasstation.info/."

[177] Zero knowledge set membership implementation. [Online]. Available: https://github.com/ing-bank/zkproofs

[178] H. Kellerer, U. Pferschy, and D. Pisinger, "Knapsack problems," *Springer, Berlin*, 2004.

[179] D. Chaum, "Blind signatures for untraceable payments," *Proc. of Springer International conference of Advances in cryptology, Boston, MA*, pp. 199–203, 1983.

[180] T. Okamoto, "Efficient blind and partially blind signatures without random oracles," *Proc. of Springer International Conference of Theory of Cryptography*, pp. 80–99, 2006.

[181] D. Chaum, "Blind signatures for untraceable payments," *Proc. Advances in cryptology (CRYPTO)*, pp. 199–203, 1983.

[182] Q. ShenTu and J. Yu, "A blind-mixing scheme for bitcoin based on an elliptic curve cryptography blind digital signature algorithm," *CoRR*, vol. abs/1510.05833, 2015. [Online]. Available: http://arxiv.org/abs/1510.05833

[183] C. Schnorr, "Efficient signature generation by smart cards," *J. Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

[184] R. Gennaro, D. Leigh, R. Sundaram, and W. S. Yerazunis, "Batching schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices," *proc. of International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pp. 276–292, 2004.

[185] Y. Zhou, M. Wang, H. Hao, L. Johnson, and H. Wang, "Plug-in electric vehicle market penetration and incentives: a global review," *Mitigation and Adaptation Strategies for Global Change*, vol. 20, no. 5, pp. 777–795, 2015.

[186] N. Saxena, S. Grijalva, V. Chukwuka, and A. V. Vasilakos, "Network security and privacy challenges in smart vehicle-to-grid," *IEEE Wireless Communications*, vol. 24, no. 4, pp. 88–98, 2017.

[187] Ganache. [Online]. Available: https://www.trufflesuite.com/ganache

[188] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for c: Verifying program executions succinctly and in zero knowledge," *Proc. of the annual cryptology conference*, pp. 90–108, 2013.

[189] M. Raya and J.-P. Hubaux, "Securing vehicular ad hoc networks," *Journal of computer security*, vol. 15, no. 1, pp. 39–68, 2007.

VITA

Mohamed Baza is currently a PhD candidate in Department of Electrical Computer Engineering, Tennessee Technological University, TN, USA. He works under the supervision of Dr. Mohamed Mahmoud. He received his BS and MS in Electrical and Computer Engineering from Benha University, Cairo, Egypt. His research interests include several topics in the area of cybersecurity, including blockchains, secure decentralized applications, machine learning, smart grids, and VANETs. He has more than two years of industry experience in information security field in Apache-Khalda Petroleum Company. He has received the Technical Committee's Best Paper Award in IEEE International Conference on Smart Applications, Communications and Networking (SmartNets 2019). He is an active reviewer in Top IEEE journals/conferences.