COMP4240 Special Topic G

Blockchain Based Secure Car Sharing System

Final Report

Zaw Moe Htat (c3193528)

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Abstract

Blockchain is the new type of technology that copy the block and store the data, then link it with the previous one. It has been used in many other applications for decentralisation and security. When it comes to development of distributed system, the security issues are the main concern in many factors. Fortunately, with the advantage of immutability of the Blockchain nature, the data can be kept securely without needing to concern of being altered by someone. The main objective of this project is to design and build the secure distributed system based on Blockchain that is used for car sharing service. In such a large distributed system, it is important to securely verify the users. Our approach is using ECDSA algorithm to verify the signature of the users within the Blockchain. The system uses cloud database for massive data storage solution.

Contents

1 Introduction **8**

1.1 Background 8

2 Project Overview **9**

2.1 Aims 9

2.2 Challenges 9

2.3 Scope 9

2.4 Outcomes 10

2.5 Deliverables 10

3 Approach/Methodology **12**

3.1 Signature Verification with Blockchain 12

3.1.1 Methods 13

3.1.2 Assumptions 14

3.1.3 Requirements 14

3.1.4 Constraints 14

3.2 RESTful APIs 14

3.2.1 Methods 15

3.2.2 Assumptions 15

3.2.3 Requirements . . . 15

3.2.4 Constraints . . . 16

4 System Architecture and Design **17**

4.1 Overview System Diagram . . . 17

4.2 System Components . . . 17

5 Scenarios **19**

6 Implementation **20**

6.1 Class Diagrams . . . 20

6.2 Demonstration of Scenarios . . . 21

6.3 Screenshots of Example Walk Through . . . 22

6.3.1 New user registration . . . 22

6.3.2 Owner adding a car . . . 22

6.3.3 Borrower booking a car . . . 24

6.3.4 Borrower returning a car . . . 24

7 Discussion **26**

7.1 System Improvements/Refinements . . . 26

7.2 Security Requirements and Extensions . . . 26

8 Conclusion **28**

8.1 Future work 28

Appendices **30**

A Solidity implementation of CarNet **30**

B JavaScript implementation of CarNet **34**

List of Figures

1 Flow diagram of signature verification 13

2 Overview system architecture 17

3 Components of the system. 18

4 Class Diagram 20

5 User registration demonstration 22

6 Owner adding car demonstration 23

7 Blockchain transaction results of adding car. 23

8 Blockchain transaction results of booking car. 24

9 List of rented cars. 24

10 Blockchain transaction results of returning car. 25

11 List of rented cars. 25

List of Tables

1 Deliverables of the project 11

2 Demonstration of scenarios 21

1 Introduction

1.1 Background

Car sharing services are growing dramatically lately due to its convenience and efficiency in cost. Car sharing systems allows commuters to choose the type of cars they want to drive. With that system, car owners can list their car to rent on cloud server where users can choose the car online and book the car for a certain period. The system that provides such highly distributed service must be secured as some sensitive data need to be exchanged such as the location of the vehicle, keys to unlock the car, and payment details of the user. In this project, the Blockchain network architecture is used in the system for secure communication. Blockchain is a distributed ledger that cryptographically links its data in such a way that the data becomes unmodifiable without modifying all previous data linked in the chain. All transactions made from users to the cloud are verified by blockchain network domain. Thus, users can choose the car and update the details securely.

2 Project Overview

This project involves Blockchain technology and API development. In this project, the car sharing system uses Blockchain network to verify users by their signatures and cloud database to store all information of users and car details.

2.1 Aims

The aims of this project are:

- To create a system that verifies the transaction of the user by using the signature in the Blockchain network
- To successfully and securely add a car, book a car and return a car
- To develop an API for furture application development

2.2 Challenges

There are several challenges for this project such as development time frame and understanding the Blockchain technology. The main challenges, however, are the implementation of Blockchain into the system. There are several ways to implement Blockchain application, however they are mostly divided by their implementation with via different programming language, most created specifically for block chaining. With the variety of blockchain types available, the programmers creating these different types tend to lean towards specific goals for their own use, and as such may make it more difficult to fit it into our own purposes.

2.3 Scope

Due to time constraints and general unfamiliarity with Blockchain networks, it has been decided that the scope of the project will be kept narrow and will be expanded as each goal is met if time permits.

The initial scope for the project is the implementation of the Blockchain network, with signature verification.

The first scope expansion is developing API to allow renter users to book vehicles for use and allow owner users to add cars onto the service.

The next expansion of the scope is including a payment system within the service.

2.4 Outcomes

The outcome is to primarily build a Blockchain network that can support the secure car sharing service.

Users are expected to be able to add vehicles, view all available vehicles and book a vehicle for rent from their application. Each user needs to sign a request, which is then passed into Blockchain network for verification. Users are not expected to proceed if the signature is not belong to the user who sends the request.

2.5 Deliverables

The main deliverables of this project are as described in table 1.

No.	Deliverables	Priority	Description
1	Blockchain Network	High	Implementation of Blockchain network that verifies the signature of users
2	Users API	High	API for Users reigstraion and login
3	Cars API	High	API to add car, book car, return car, list cars
4	Application	Medium	Interactive application develoment using API
5	Payment	Low	Payment service on car returned

Table 1: Deliverables of the project

3 Approach/Methodology

In this project, there are two different infrastructures to be considered in our approach, which are cloud infrastructure and Blockchain infrastructure. In cloud server, there is a storage that stores all data and resources. When the data is updated or added in the cloud storage, the user needs to sign the transaction which is then stored in the block and broadcast the block into the Blockchain network for signature verification. All those services are performed by calling APIs in the cloud. The detail approach is discussed in the following.

3.1 Signature Verification with Blockchain

The user who updates or adds the data into the database, signs the data with the private key. The signature signed to that particular data is stored in the block. The block is then added into the Blockchain network. When the the data is accessed and modified, the user is verified by the signature stored in the Blockchain. If the signature is not belong to the user who requests for the transaction, the block will be rejected. Thus, the data will not be modified or altered.

In our Blockchain architecture, the block stores two signatures. One signature for the owner and another one for the borrower. When the vehicle is added onto the server, the owner signs the vehicle information and stores it in the Blockchain. When the borrower rents the car, it validates the car to make sure that car is belong to the right owner by using the signature that the owner signed. If it is valid, the signature of borrower will be generated and stored it in the Blockchain. On issuing the vehicle return request, the signature of both owner and borrower are validated and the process will continue. The structure of this transaction can be described as:

$$TID_{Current}||TID_{Previous}||PK_{Owner}||Sig_{Owner}||PK_{Borrower}||Sig_{Borrower}$$

where, TID is transaction ID, PK is public key and Sig is signature. The figure 1 shows the flow of signature verification on three events: adding vehicle, booking vehicle and returning vehicle.

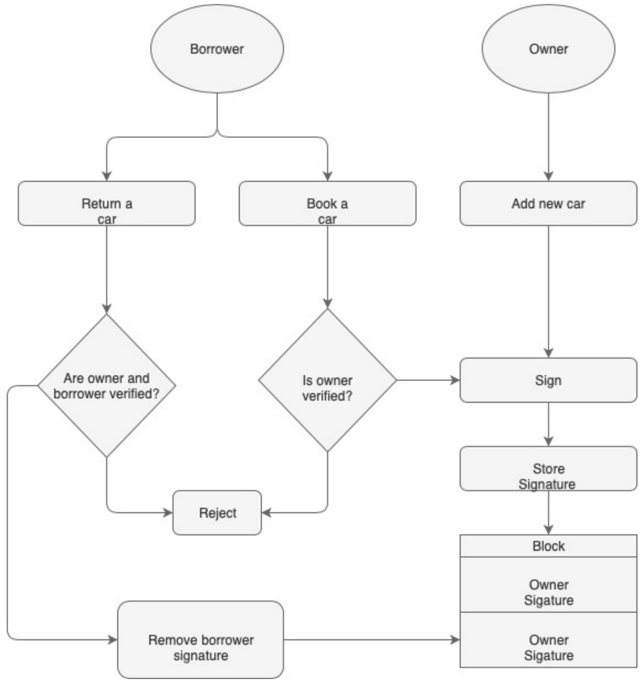


Figure 1: Flow diagram of signature verification

3.1.1 Methods

Our Blockchain is implemented in Solidity [4], Ethereum smart contract programming language, on Ethereum Blockchain network [2]. Solidity is one of the most powerful programming language for Ethereum based Blockchain applications. It has the JavaScript API for client side to communciate with Blockchain network. There are several methods that can be used to generate and verify signatures. As the security level is concerned, our implementation for signatures verification is by following Elliptic Curve Digital Signature Algorithm (ECDSA) [3], which has higher level of security compare to other algorithms such as RSA. The algorithm to generate the signature is as following:

$$sig = sign(sha3(H_c, pk), sk) \tag{1}$$

where H_c is unique hash code of car data, pk is public key and sk is private key. The public key can be recovered from the signature by using the following algorithm:

$$pk = R(H, v, r, s) \tag{2}$$

where H is hashed message, v, r, and s are ECDSA parameter values extracted from signature.

3.1.2 Assumptions

It is assumed that all users who are using the service has valid Ethereum accounts.

3.1.3 Requirements

In order to run the system, Ethereum Blockchain network is required. The network can be either Ethereum Testnet or local Ethereum Virtual Network. All users are required to be registered and authenticated in order to send the transaction to the Blockchain network. If the Ethereum network is running on local machine, the Ethereum account of the user must be created or exist in the that network.

3.1.4 Constraints

The only constrain in the development of Blockchain with Solidity is the limitation of the language and lack of the nature of traditional programming languages in many ways.

3.2 RESTful APIs

Our cloud system is REST based system which provides RESTful APIs for end-to-end client-server communication. The APIs are called to perform tasks such as user authentication, getting list of available cars, adding cars onto the server, issuing requests for cars booking and returning cars. The APIs communicate with cloud database and Blockchain network directly.

The details of how cloud server communicates with Blockchain network will be discussed in System Architecture and Design section.

3.2.1 Methods

In this project, RESTful APIs are developed in Nodejs. The choice of using JavaScript for server instead of other programming languages because of the Solidity JavaScript API, called Web3js, which provides the better development for Blockchain based application like our project. Using Web3js, it allows data to be sent to Blockchain network by deploying the smart contract. Our cloud server uses MySQL database to store the data.

When the user called the APIs that needs to modify the data in the database, the user will be verified on the Blockchain. The modification will only be made if the user is verified.

3.2.2 Assumptions

There are three assumptions made for RESTful APIs calls which are as following:

- All API requests are authorised.
- All data sent to the server are valid and are not malicious.
- The server can handle unlimited APIs called.

3.2.3 Requirements

There are two main requirements for this server to set up, which are described as following:

- Nodejs is required to be installed on the server since the server is implemented in Nodejs.
- Ethereum server is required to be running at the time when the server is up.

3.2.4 Constraints

In this project, time constraint is the only constraint for this API development. Most of the APIs for basic features such as authenticating users, adding a car, booking a car, returning a car and listing cars, have been implemented. Giving the limited amount of time, the entire full system has not been implemented completely which includes payment system and complete application to interact with the system.

4 System Architecture and Design

In this section, we explain how components in the system communicate with each other.

4.1 Overview System Diagram

Our system consists of three different servers for three different purposes as shown in Figure 2. Web/app server mainly serves for front-end application such as web application and mobile application. The application sends requests via APIs to the cloud server. The cloud server communicates with Ethereum Blockchain network by using the web3js Ethereum APIs. The responses are sent back to client correctly on each request.

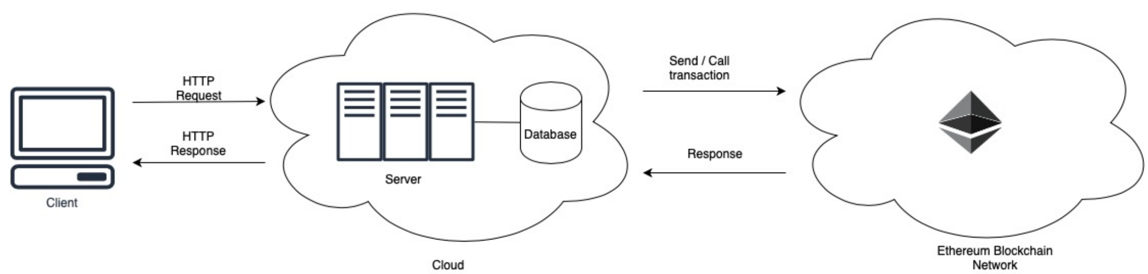


Figure 2: Overview system architecture

4.2 System Components

In the cloud server, the middleware component filter out HTTP requests from the clients for security level to prevent any unauthorised access or attacks. The controllers perform the action requested by the client such as user authentication and cars management. The controller talks to Blockchain network via CarNet component which is the implementation of Ethereum client. The CarNet is responsible for signing the data and send it to Blockchain for verification. The database component is used by the controller to store and update data. The Figure 3 shows the details of how each component in the system are related.

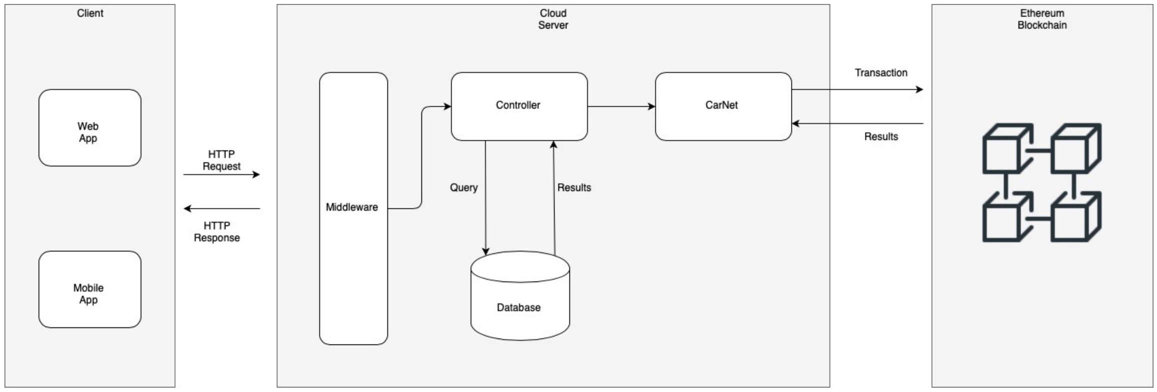


Figure 3: Components of the system.

5 Scenarios

In this project, there are several major use cases, which are as described as below.

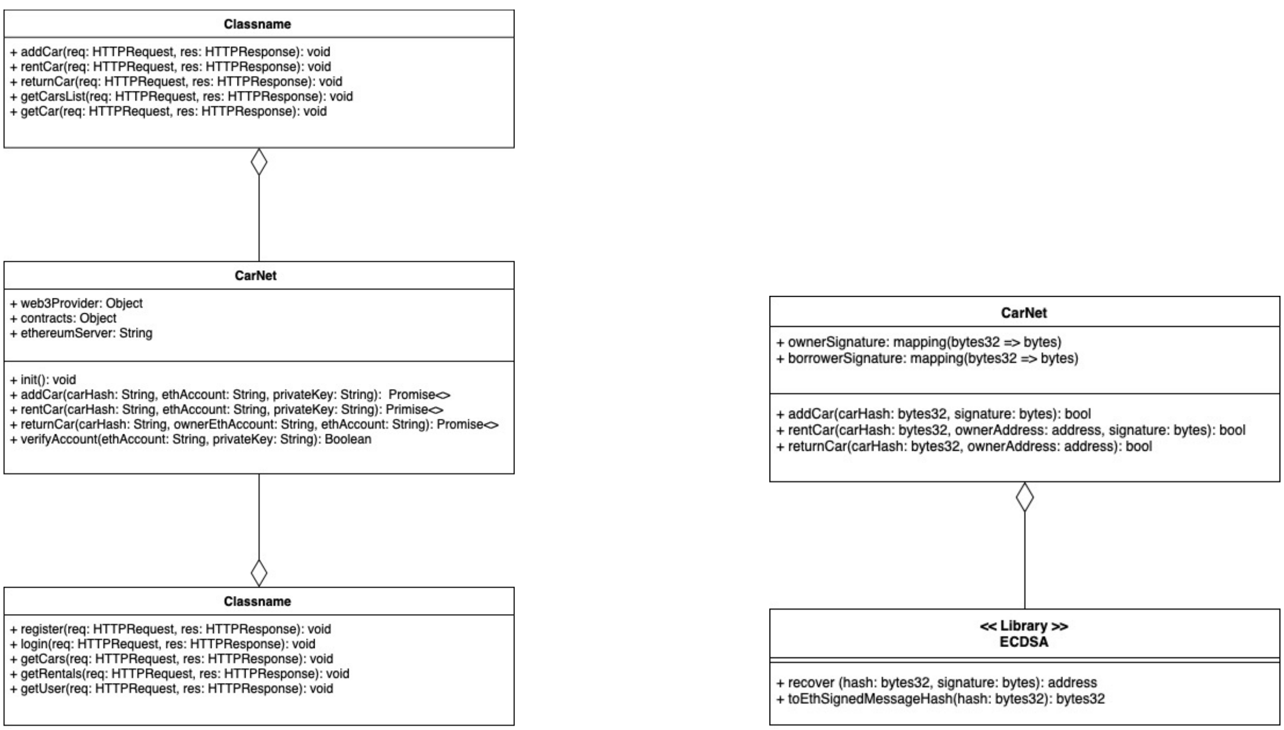
1. A new user registering an account
2. An owner adding a car
3. A borrower booking a car
4. A borrower returning a car

6 Implementation

In this section, we describe the implementation of the system by showing class diagrams and walk through examples with screen shots.

6.1 Class Diagrams

The implementation consists of two parts: JavaScript and Solidity. APIs are implemented in JavaScript as shown in figure 4a and smart contract for Ethereum Blockchain network is implemented in Solidity as shown in fighre 4b. JavaScript CarNet uses web3js [6] and Truffle [5] library to deploy the CarNet smart contract and call the methods from it (see Appendix B).



(a) JavaScript API Class Diagram.

(b) Solidity Class Diagram.

Figure 4: Class Diagram

6.2 Demonstration of Scenarios

The scenarios mentioned above are demonstrated as shown in Table 2.

No.	Scenario	Description of demonstration
1	New user registration	The following functionalities will be demonstrated: • User signing up for listing a car. • User signing up for borrowing a car.
2	Owner adding a car	The following functionalities will be demonstrated: • Owner get the Blockchain transaction results on success of adding. • Owner see the list of added cars.
3	Borrower booking a car	The following functionalities will be demonstrated: • Borrower get the Blockchain transaction results on success of booking. • Borrower see the list of rented cars.
4	Borrower returning a car	The following functionalities will be demonstrated: • Borrower get the Blockchain transaction results on success of returning. • Borrower does not see the returned car in the rental list.

Table 2: Demonstration of scenarios

6.3 Screenshots of Example Walk Through

In order to demonstrate the API calls, Postman is used to perform it.

6.3.1 New user registration

The figure 5a and 5c shows the inputs for registering new user account and figure 5b and 5d shows the response when the registration is successfully completed.

KEY	VALUE
first_name	Jame
last_name	Smith
email	jame.smith@gmail.com
password	12345678
confirm_password	12345678
user_type	owner
ethereum_address	0x7C252EFedBC68C903D97aAa2e7e74F5D5cbc010d
ethereum_private_key	0bad45792014bf405060a93648f2606d585fc2fc3d73c324e2ab5216e5335efc

(a) Inputs for owner type user registration.

```
"data": {
    "id": "4",
    "email": "jame.smith@gmail.com",
    "eth_account": "0x7C252EFedBC68C903D97aAa2e7e74F5D5cbc010d"
}
```

(b) The response from the server on success of owner user registration.

KEY	VALUE
first_name	Scott
last_name	Johnson
email	scott.johnson@gmail.com
password	87654321
confirm_password	87654321
user_type	borrower
ethereum_address	0x344d14d6f4e6568FEDC42CD0744C24f1Ca629512
ethereum_private_key	dd1c65ce74e262793fb4b5b61d6092b3ed832658f0928d316ce587808ebb3115

(c) Inputs for borrower type user registration.

```
"data": {
    "id": "5",
    "email": "scott.johnson@gmail.com",
    "eth_account": "0x344d14d6f4e6568FEDC42CD0744C24f1Ca629512"
}
```

(d) The response from the server on success of borrower user registration.

Figure 5: User registration demonstration

6.3.2 Owner adding a car

The figure 6a shows the inputs for adding the car and figure 6b shows the list of added car. When the car is added successfully, the signature is generated and added to the Blockchain. The transaction of the Blockchain is returned(see Figure 7) when the block has been added into the Blockchain network successfully.

KEY	VALUE
make	Honda
model	Civic
transmission	automatic
num_seat	5
rego	HC33AC
price	50
address	University Dr, Callaghan NSW 2308
status	available
last_service	2019-10-01
available_from	2019-10-20
available_to	2019-11-03

(a) Inputs for adding car to the server.


```
"data": [
    {
        "id": 3,
        "make": "Honda",
        "model": "Civic",
        "transmission": "automatic",
        "num_seat": 5,
        "rego": "HC33AC",
        "price": 50,
        "address": "University Dr, Callaghan NSW 2308",
        "status": "available",
        "user_id": 4,
        "last_service": "2019-09-30T14:00:00.000Z",
        "available_from": "2019-10-19T13:00:00.000Z",
        "available_to": "2019-11-02T13:00:00.000Z",
        "hash": "c1fc519dbe259dcfe605b3f9385453af",
        "created_at": "2019-10-20T13:00:00.000Z"
    },
    {
        "id": 4,
        "make": "Toyota",
        "model": "Corolla",
        "transmission": "automatic",
        "num_seat": 5,
        "rego": "DB77HR",
        "price": 32,
        "address": "University Dr, Callaghan NSW 2308",
        "status": "available",
        "user_id": 4,
        "last_service": "2019-09-30T14:00:00.000Z",
        "available_from": "2019-10-19T13:00:00.000Z",
        "available_to": "2019-11-02T13:00:00.000Z",
        "hash": "b3c2a7ca12dd9274c65e84ba97470e45",
        "created_at": "2019-10-20T13:00:00.000Z"
    }
]
```


(b) List of added car.

Figure 6: Owner adding car demonstration


```
"data": {
    "transaction": {
        "tx": "0x7698da96bf893c60708f09b7b625404974cc19b5717553927eb354a06e7f0c3b",
        "receipt": {
            "transactionHash": "0x7698da96bf893c60708f09b7b625404974cc19b5717553927eb354a06e7f0c3b",
            "transactionIndex": 0,
            "blockHash": "0x087411fbcf3f81aa224ef8fd6f4d8bdc7c9513a20007a15d277c848cc6a49c55",
            "blockNumber": 19,
            "from": "0x7c252efedbc68c903d97aaa2e7e74f5d5cbc010d",
            "to": "0x03bf5ded23fcf993076816ed3c0881aa287c4987",
            "gasUsed": 114508,
            "cumulativeGasUsed": 114508,
            "contractAddress": null,
            "logs": [],
            "status": true,
            "logsBloom":
                "0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
                000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
                000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
                000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
                000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
                00000000000000000000000000000000000000000000000000000000000000000",
            "v": "0x1c",
            "r": "0xdcadfd8e9b2001b5caf9557e81644d29e74861bd3213336491dbb578fa92c6a3",
            "s": "0xee3d3d4ba96424f8ae46bf638054c023bd69b98dad93536b5dbbda80f13674",
            "rawLogs": []
        },
        "logs": []
    }
}
```


Figure 7: Blockchain transaction results of adding car.

6.3.3 Borrower booking a car

The figure 10 shows the transaction of Blockchain when the car is successfully booked and figure 9 shows the list of car that is currently booked by the borrower.


```
"data": {
    "tx": "0xcefeba49cab43b10437b6b6162f46523544083f745ff8c3f78ab56e54512af8d",
    "receipt": {
        "transactionHash": "0xcefeba49cab43b10437b6b6162f46523544083f745ff8c3f78ab56e54512af8d",
        "transactionIndex": 0,
        "blockHash": "0x62a3732f575a926ac646bd7d7dbb1517cf48d5409b1b81c206092de75f62cffb",
        "blockNumber": 21,
        "from": "0x344d14d6f4e6568fedc42cd0744c24f1ca629512",
        "to": "0x03bf5ded23fcf993076816ed3c0881aa287c4987",
        "gasUsed": 122121,
        "cumulativeGasUsed": 122121,
        "contractAddress": null,
        "logs": [],
        "status": true,
        "logsBloom":
            "0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            000000000000000000000000000000000000000000000",
        "v": "0x1b",
        "r": "0xafbdbcccf29d65e3f9af044af512720b375d92c835c694c88d143af3c9845c95",
        "s": "0x3779ab84d25d8745b29b5f01d77b7a7c46b8176566e7fb65a47be1f8fecf81e5",
        "rawLogs": []
    },
    "logs": []
}
```


Figure 8: Blockchain transaction results of booking car.


```
"data": [
    {
        "id": 1,
        "make": "Toyota",
        "model": "Corolla",
        "transmission": "automatic",
        "num_seat": 5,
        "rego": "BR45DB",
        "price": 25,
        "address": "Somewhere, Newcastle 2222",
        "status": "unavailable",
        "user_id": 1,
        "last_service": "2019-09-30T14:00:00.000Z",
        "available_from": "2019-10-19T13:00:00.000Z",
        "available_to": "2019-11-02T13:00:00.000Z",
        "hash": "8fd9e69233ea8fdbf9ee2a290a094bab",
        "created_at": "2019-10-18T13:00:00.000Z",
        "rental_id": 5
    },
    {
        "id": 3,
        "make": "Honda",
        "model": "Civic",
        "transmission": "automatic",
        "num_seat": 5,
        "rego": "HC33AC",
        "price": 50,
        "address": "University Dr, Callaghan NSW 2308",
        "status": "unavailable",
        "user_id": 4,
        "last_service": "2019-09-30T14:00:00.000Z",
        "available_from": "2019-10-19T13:00:00.000Z",
        "available_to": "2019-11-02T13:00:00.000Z",
        "hash": "c1fc519dbe259dcfe605b3f9385453af",
        "created_at": "2019-10-20T13:00:00.000Z",
        "rental_id": 6
    }
]
```


Figure 9: List of rented cars.

6.3.4 Borrower returning a car

The figure 10 shows the transaction of Blockchain when the car is successfully returned. As compare the figure 9, the figure 11 shows the car ID 3 is gone as a result of car being returned.

```
"data": {
    "tx": "0x4f1629607c7e9662c1efb9370abad8b07bd2d747143248066df51e0166383ad6",
    "receipt": {
        "transactionHash": "0x4f1629607c7e9662c1efb9370abad8b07bd2d747143248066df51e0166383ad6",
        "transactionIndex": 0,
        "blockHash": "0x2610fdc4abb0c9fdf866896bdf80e278b0d5772793ef239b85a1cfad82a96167",
        "blockNumber": 23,
        "from": "0x344d14d6f4e6568fedc42cd0744c24f1ca629512",
        "to": "0x03bf5ded23fcf993076816ed3c0881aa287c4987",
        "gasUsed": 29086,
        "cumulativeGasUsed": 29086,
        "contractAddress": null,
        "logs": [],
        "status": true,
        "logsBloom":
            "0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            000000000000000000000000000000000000000000000",
        "v": "0x1b",
        "r": "0xfdec5ff1543f47fdf293864e547bfda08e0eed67af067cc9113068af9b3c4a59",
        "s": "0x1371c0d03c73fa763daa6e909a8778ace57af63047af295cd9a99fa8cd9a6e33",
        "rawLogs": []
    },
    "logs": []
}
```


Figure 10: Blockchain transaction results of returning car.


```
"data": [
    {
        "id": 1,
        "make": "Toyota",
        "model": "Corolla",
        "transmission": "automatic",
        "num_seat": 5,
        "rego": "BR45DB",
        "price": 25,
        "address": "Somewhere, Newcastle 2222",
        "status": "unavailable",
        "user_id": 1,
        "last_service": "2019-09-30T14:00:00.000Z",
        "available_from": "2019-10-19T13:00:00.000Z",
        "available_to": "2019-11-02T13:00:00.000Z",
        "hash": "8fd9e69233ea8fdbf9ee2a290a094bab",
        "created_at": "2019-10-18T13:00:00.000Z",
        "rental_id": 5
    }
]
```


Figure 11: List of rented cars.

7 Discussion

In this section, we discuss system improvements and refinements, and security requirements and extensions.

7.1 System Improvements/Refinements

The current system has only been implemented with basic features for car sharing service. There are a few key features left to be implemented. The system could be improved by adding payment system into it. The payment system could be implemented in many ways. Traditionally, the credit card payment methods could be used as a payment options such as Paypal, MastCard, Visa and so on. However, with the Blockchain infrastructure, the crypto currency could be considered as a payment options. Since our Blockchain is based on Ethereum, we could improve the system to implement ERC-20 token, which could be used as crypto currency for our car sharing service.

Although this project uses Blockchain, it is not the best application of it. As high as 92% of Blockchain businesses fail [1], and one of the primary factors was that Blockchain was used inappropriately where simple database or end-to-end communication would have sufficed. Blockchain requires multiple stakeholders willing to invest resources into verifying blocks, which this project does not have, and as a result is likely to be unsuccessful if deployed to market. However, if changed are made to reconsider car sharing as the focus of the project, it may be more viable. The main factor is that the Blockchain network provides immutable history of the car's history, which may be useful for insurers or when reselling a car. For these reasons there is a possibility to 'genericise' the project in the future to provide just automotive history rather than car sharing, which would likely create a more viable business.

7.2 Security Requirements and Extensions

The communication between both the user and supplier will require confidential information on both parties such as names, locations, and other similar information. This means that these communications must be passed securely between the two members. Since the Blockchain is public, the passed data must not be handled within the Blockchain or should be suitably encrypted. All API calls to the cloud server are required to be secure by issuing authentication token.

However, future payment system may also want to have to access third party payment services (e.g. PayPal, Mastercard, etc.). For these services, permission must be sought from those companies and, if approved, additional changes must be made to ensure security and access is compliant with the services API.

8 Conclusion

In this report, we have presented the solution for secure car sharing service using Blockchain technology for signature verification.

For Blockchain application, we implemented in Ethereum smart contract, Solidity, on Ethereum Blockchain network. We use ECDSA algorithm for secure signature generation and verification.

The Blockchain functionalities are called from the cloud when the APIs are requested. All APIs are deployed in the cloud server, which are implemented in Nodejs. As a result, we have implemented the basic functioning application which can be used for user registration and login, adding a car, viewing the car list, booking a car and returning a car when it is done using.

8.1 Future work

Future work will investigate in payment system, more features and functionalities in RESTful APIs, and front-end application development such as web application and mobile application, with better user interface and experience.

References

[1] D. Disparte. *Why Enterprise Blockchain Projects Fail*. Forbes. 20 May 2019. [Online]. Available: `https://www.forbes.com/sites/dantedisparte/2019/05/20/why-enterprise-blockchain-projects-fail`. [Accessed 16 October 2019].

[2] ethereum.org. (2019). Home | Ethereum. [online] Available at: `https://www.ethereum.org` [Accessed 20 Oct. 2019].

[3] Johnson, D., Menezes, A. and Vanstone, S. (2001). *The Elliptic Curve Digital Signature Algorithm (ECDSA)*. International Journal of Information Security, 1(1), pp.36-63.

[4] Solidity.readthedocs.io. (2019). *Solidity — Solidity 0.5.12 documentation*. [online] Available at: `https://solidity.readthedocs.io/en/v0.5.12/` [Accessed 20 Oct. 2019].

[5] Truffle Suite. (2019). *Truffle | Truffle Suite*. [online] Available at: `https://www.trufflesuite.com/truffle` [Accessed 20 Oct. 2019].

[6] Web3js.readthedocs.io. (2019). *web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation*. [online] Available at: `https://web3js.readthedocs.io/en/v1.2.1/` [Accessed 20 Oct. 2019]

Appendices

A Solidity implementation of CarNet

```
pragma solidity ^0.5.0;

import "./ECDSA.sol";

/// @title  The blockchain network for car owner validation
/// @author Kelvin Yin
/// @notice Use this contract for car ownership validation
/// @dev    Each car will have unique hash which is then map to
///         the address of the owner/borrower.
contract CarNet {

  // Store the signature of the car owner
  // [CarHash => OwnerSignature]
  mapping(bytes32 => bytes) ownerSignature;

  // Store the signature of the car borrower
  // [CarHash => BorrowerSignature]
  mapping(bytes32 => bytes) borrowerSignature;

  /// @notice Add new block on when car is added
  /// @dev    Make sure the message sender is the owner of the car
  ///
  /// @param  carHash   Unique hash code for the car
  /// @param  signature Signature that was signed by the owner of
     the car
  ///
  /// @return  True if success, throw error otherwise
```

```
function addCar(bytes32 carHash, bytes memory signature) public
    returns (bool) {

  // Get signer from signature
  address signer = ECDSA.recover(
    ECDSA.toEthSignedMessageHash(
      keccak256(abi.encodePacked(carHash, msg.sender))
    ), signature
  );

  // Verify if the sender is the owner
  require(
    signer == msg.sender,
    "Unauthorised signer has been dectected."
  );

  // Add owner signature
  ownerSignature[carHash] = signature;

  return true;

}

/// @notice Add new block when car is rented
/// @dev    Make sure the message sender is the borrower who
    borrow the car
///
/// @param  carHash      Unique hash code for the car
/// @param  ownerAddress Address of car owner
/// @param  signature    Signature theat was signed by the
    borrower of the car
///
/// @return True if success, throw error otherwise
```

```
function rentCar(bytes32 carHash, address ownerAddress, bytes
  memory signature) public returns (bool) {

  // Get address from owner signature
  address _ownerAddress = ECDSA.recover(
    ECDSA.toEthSignedMessageHash(
      keccak256(abi.encodePacked(carHash, ownerAddress))
    ), ownerSignature[carHash]
  );

  // Verify the car belong to the right owner
  require(
    _ownerAddress == ownerAddress,
    "Car does not belong to the owner."
  );

  // Get signer from signature
  address signer = ECDSA.recover(
    ECDSA.toEthSignedMessageHash(
      keccak256(abi.encodePacked(carHash, msg.sender))
    ), signature
  );

  // Verify the borrower address
  require(
    signer == msg.sender,
    "Unauthorised signer has been dectected."
  );

  // Add borrower signature
  borrowerSignature[carHash] = signature;

  return true;
```

```
}

/// @notice Add new block when car is returned
/// @dev     Make sure the message sender is the borrower who
    borrow the car
///
/// @param  carHash       Unique hash code for the car
/// @param  ownerAddress Address of car owner
///
/// @return True if success, throw error otherwise
function returnCar(bytes32 carHash, address ownerAddress) public
    returns (bool) {

  // Get address from owner signature
  address _ownerAddress = ECDSA.recover(
    ECDSA.toEthSignedMessageHash(
      keccak256(abi.encodePacked(carHash, ownerAddress))
    ), ownerSignature[carHash]
  );

  // Verify the car belong to the right owner
  require(
    _ownerAddress == ownerAddress,
    "Car does not belong to the owner."
  );

  // Get address from borrower signature
  address _borrowerAddress = ECDSA.recover(
    ECDSA.toEthSignedMessageHash(
      keccak256(abi.encodePacked(carHash, msg.sender))
    ), borrowerSignature[carHash]
  );
```

```

    // Verify the car returning belong to the right borrower
    require(
      _borrowerAddress == msg.sender,
      "Car does not belong to the borrower."
    );

    // Delete the borrower signature for the car
    delete borrowerSignature[carHash];

    return true;
  }

}
```

B JavaScript implementation of CarNet

```
/**
 * CarNet JavaScript library for CarNet contract
 *
 * @author Kelvin Yin
 */

const Web3            = require('web3');
const fs              = require('fs');
const TruffleContract = require('@truffle/contract');
const ethereumjsAbi   = require('ethereumjs-abi');
const truffleConfig   = require('../truffle-config');

module.exports = {
    web3Provider : null,
    contracts : {},
```

```
    ethereumServer: 'http://' + truffleConfig.networks.development.
        host + ':' + truffleConfig.networks.development.port,

    /**
     * Initialise CarNet contract
     *
     * This method initialise web3 provider and and contract
     */
    init: function() {

        // Check if web3 has already been provided
        if (typeof web3 !== 'undefined') {
            this.web3Provider = web3.currentProvider;
        } else {
            this.web3Provider = new Web3.providers.HttpProvider(this
                .ethereumServer);
        }

        // Create web3 object to connect to blockchain
        web3 = new Web3(this.web3Provider);

        // Get artifact from CarNet contract
        var carNetArtifact = JSON.parse(fs.readFileSync('blockchain/
            build/contracts/CarNet.json'));

        // Add contract
        this.contracts.CarNet = TruffleContract(carNetArtifact);
        this.contracts.CarNet.setProvider(this.web3Provider);

    },

    /**
     * Add car into blockchain.
```

```
     *
     * @param {string} carHash     Unique hash code for car to be added.
     * @param {string} ethAccount The ethereum address of the car owner.
     * @param {string} privateKey Private key of the ethereum address.
     *
     * @returns Contract transaction.
     */
    addCar: async function(carHash, ethAccount, privateKey) {

        // Deploy contract
        let instance = await this.contracts.CarNet.deployed();

        // Hash the message
        const hash = '0x' + ethereumjsAbi.soliditySHA3(
            ['bytes32', 'address'],
            [carHash, ethAccount]
        ).toString('hex');

        // Sign the hash message
        const signedHash = web3.eth.accounts.sign(hash, privateKey);

        // Get the signature
        const signature = signedHash.signature;

        return await instance.addCar(
            web3.utils.fromAscii(carHash),
            signature,
            {
                from: ethAccount,
                gas: 3000000
```

```
                }
            );

    },

    /**
     * Add rent car information into block chain
     *
     * @param {string} carHash          Unique hash code for car
     * @param {string} ownerEthAccount  Ethereum address of the owner
     * @param {string} ethAccount       Ethereum address of the borrower
     * @param {string} privateKey       Private key of borrower ethereum address
     *
     * @return Contract transaction.
     */
    rentCar: async function(carHash, ownerEthAccount, ethAccount, privateKey) {

        // Deploy contract
        let instance = await this.contracts.CarNet.deployed();

        // Hash the message
        const hash = "0x" + ethereumjsAbi.soliditySHA3(
            ['bytes32', 'address'],
            [carHash, ethAccount]
        ).toString('hex');

        // Sign the hash mesasge
        const signedHash = web3.eth.accounts.sign(hash, privateKey);

```

```
        // Get the signature
        const signature = signedHash.signature;

        return await instance.rentCar(
            web3.utils.fromAscii(carHash),
            ownerEthAccount,
            signature,
            {
                from: ethAccount,
                gas: 300000
            }
        );
    },

    /**
     * Add return car information into blockchain
     *
     * @param {string} carHash           Unique hash code for car
     * @param {string} ownerEthAccount  Ethereum address of car
        owner
     * @param {string} ethAccount       Ethereum address of car
        borrower
     */
    returnCar: async function(carHash, ownerEthAccount, ethAccount)
        {

        // Deploy contract
        let instance = await this.contracts.CarNet.deployed();
        return await instance.returnCar(
            web3.utils.fromAscii(carHash),
            ownerEthAccount,
            {
                from: ethAccount,
```

```
                    gas: 300000
                }
            );
    },

    /**
     * Verify the account
     *
     * @param {string} ethAccount Ethereum address of user
     * @param {string} privateKey Private key to the account
     *
     * @return True if correct, false otherwise
     */
    verifyAccount: function(ethAccount, privateKey) {

        const account = web3.eth.accounts.privateKeyToAccount(
            privateKey);
        if (account.address == ethAccount) {
            return true;
        } else {
            return false;
        }

    }
}
```