electronics

MDPI

Article

Reliable Vehicle Data Storage Using Blockchain and IPFS

Hyoeun Ye and Sejin Park *

Department of Computer Engineering, Keimyung University, Daegu 1095, Korea; yeyo0x0@gmail.com
* Correspondence: baksejin@kmu.ac.kr; Tel.: +82-53-580-5270

Abstract: As the importance of vehicle data increases, it has become very important to safely store them. However, because onboard diagnostics scanners generally used to store vehicle data are IoT devices, security and capacity issues exist to store data safely and efficiently. To address this, we propose a system that stores vehicle data safely and efficiently using blockchain and IPFS. Users can access the system through DApp, an Ethereum-distributed application, and manage their vehicle data. Various experiments have been conducted to demonstrate the superior performance of this system, and the experimental results show its advantages in terms of data-processing speed and cost.

Keywords: blockchain; IPFS; smart contract; DApp; vehicle data storage; data sharing; data encryption

Citation: Ye, H.; Park, S. Reliable Vehicle Data Storage Using Blockchain and IPFS. *Electronics* **2021**, *10*, 1130. https://doi.org/10.3390/electronics10101130

Academic Editors: Joel J. P. C. Rodrigues and Binod Vaidya

Received: 22 April 2021
Accepted: 8 May 2021
Published: 11 May 2021

1. Introduction

Autonomous driving technology is attracting considerable attention to the extent that various vehicle companies—such as Tesla, Google, and Hyundai—are releasing vehicles equipped with this technology [1]. Data collection and analysis of many vehicles are indispensable for autonomous vehicles to operate safely, and the collected data can be used to improve autonomous driving performance. The importance of recording and collecting vehicle data from autonomous vehicles has increased significantly because it helps diagnose the condition of the vehicle. In addition to autonomous vehicles, general vehicles can analyze drivers' driving habits through onboard diagnostics (OBD) [2], which collects vehicle data and diagnoses vehicle conditions to prevent failures and accidents. The insurance company can install an OBD scanner on an insured vehicle and provide a service that discounts insurance to subscribers who operate safely based on the collected vehicle data. In addition, vehicle data are being used in a wide variety of fields. However, because vehicle data are directly connected to the stability of vehicle operation, it is important to store them safely so that security problems such as unauthorized reading or manipulation of vehicle data do not occur. To solve this problem, blockchain [3]—a technology that can safely store data—is widely used. Blockchain is a decentralized system, and as it uses a method called digital signature, it is characterized by excellent data integrity and security, making it a suitable technology to solve the problem. Therefore, this study proposes a method for safely storing and processing vehicle data using a highly secure blockchain. The remainder of this paper proceeds as follows. Section 2 examines the background knowledge and blockchain issues, Section 3 discusses related research, Section 4 outlines important design content, Section 5 describes implementation results, Section 6 describes data sharing scenarios, Section 7 details the verification and evaluation of the implemented system, and Section 8 describes the conclusions and future research.

2. Background

Vehicle data refers to all data generated by a vehicle, such as mileage, speed, engine status, and battery status. Vehicle data can be checked through the OBD scanner, which collects data by attaching it to the OBD terminal mounted on the vehicle and the app connected to the OBD scanner [4]. However, the vehicle data collected in this manner have

security problems, such as data manipulation, and may compromise personal information. Blockchains are widely used as a solution to overcome these security issues.

Blockchain is a distributed ledger technology based on a peer-to-peer network, which records and manages blocks that are connected in the form of chains. Blockchain is widely used for data security, because it is difficult to hack or tamper with, and guarantees data integrity and security through digital signatures. However, there is a capacity issue in utilizing blockchain for vehicle data security. The maximum capacity that can be stored per block in the blockchain is very small, such as 1 MB in the case of Bitcoin; therefore, it is difficult to store large amounts of data [5]. In addition, a fee is required to process every transaction. However, vehicle data are produced in large amounts in a short amount of time; in fact, at Auto Mobility LA held in 2016, Intel CEO announced that the autonomous driving test vehicle would generate approximately 4 TB of data per day [6]. Therefore, storing large amounts of vehicle data in the blockchain is not only a cost burden but also a storage burden.

Therefore, this study proposes a system that safely stores and processes large amounts of data such as vehicle data by combining blockchain and Inter Planetary File System (IPFS) [7], a peer-to-peer distributed file system, to solve the blockchain capacity problem. The technical descriptions required to understand the proposed system are described below.

2.1. Smart Contract

A smart contract [8] is a contract that automatically executes transactions when certain conditions are met, and Vitalik Buterin introduced a smart contract to Ethereum [9]. Smart contracts have the advantage of being able to conclude contracts between individuals without an intermediary and to conclude various types of contracts and prevent forgery using the characteristics of the blockchain.

2.2. DApp (Decentralized Application)

DApp [10] stands for decentralized application, and is a decentralized distributed application built on a blockchain. Data are distributed and stored without a central server, and commands are executed through a smart contract. A representative platform that can be developed and operated is Ethereum, and in this study, we designed a smart contract web browser-based Ethereum DApp.

2.3. IPFS (Inter Planetary File System)

IPFS is a peer-to-peer distributed file system that connects all computing devices to the same file system. The size of data that can be stored in a block in a blockchain is approximately 1 MB; therefore, it is difficult to store a large amount of data. In this study, we propose a file system, IPFS, that allows a part of the vehicle data to be uploaded and only the hash value returned from the IPFS is stored in a smart contract.

3. Related Work

FileShare [11], a paper published in January 2020, proposed FileShare, a secure, distributed application framework for sharing files and data sources. FileShare performs user registration and authentication through DApp and manages the traceability and visibility of data through smart contracts. In addition, it overcomes the problem of centralized storage using IPFS for data storage. However, there is no explanation for releasing the permission after granting the data access right to the data requester during the data sharing process.

Blockchain-based, decentralized access control for IPFS [12] is based on blockchain and proposes Acl-IPFS, a large-capacity data storage system using IPFS. Acl-IPFS uses IPFS and Ethereum smart contracts to store files and handle access rights to files. In addition, the hash value obtained by uploading a file to IPFS is stored in the Ethereum smart contract, but the data encryption process, such as encrypting the data with the user's public key, is not included.

In this paper, integrating blockchain for data sharing and collaboration in mobile healthcare applications [13], as the value of personal health data collected through mobile and wearable devices increases, a medical system is proposed. According to the system proposed in this paper, the user collects data through a wearable device, which is synchronized with the cloud server of the user account through the app, and finally uploaded to the blockchain network. In addition, controlling access to user data allows users to protect their privacy. However, it does not contain any information regarding the data encryption.

In addition, when storing data, part of the data is encrypted with the user's public key and stored in the smart contract, and another part is uploaded to the IPFS, and the returned hash value is stored in the smart contract. By configuring the encryption process in such a complex step, the security of data encryption is strengthened.

4. Design

Here we propose a system that solves the capacity problem by creating a DApp, a decentralized application of Ethereum, a blockchain platform that can safely store vehicle data and combine IPFS with the blockchain. This section describes the structure of the system and its main processes.

4.1. System Structure

The overall system structure is as shown in Figure 1.

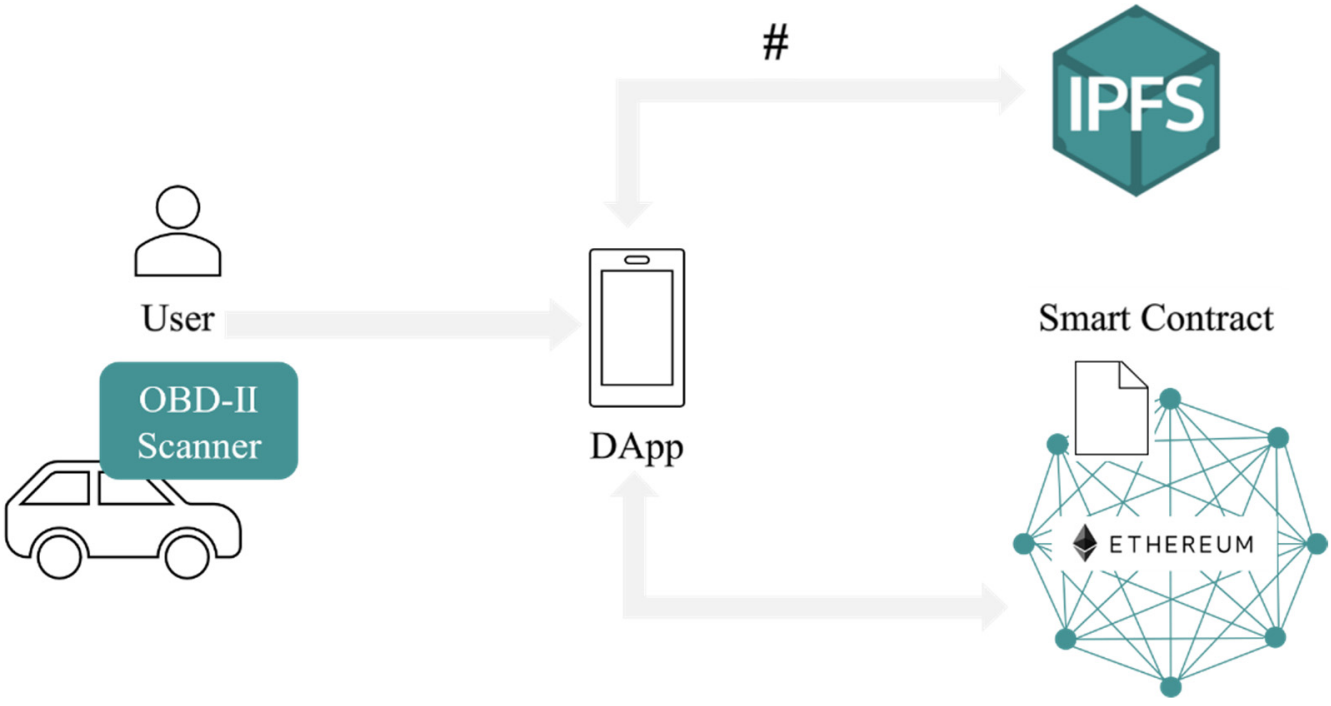

Figure 1. System architecture.

Users with vehicles equipped with OBD-II scanners receive vehicle data through their smartphones and communicate with the IPFS and blockchain through DApp. Users can subscribe to store and manage data, and the registered information is stored in an Ethereum smart contract that runs automatically when conditions are met through DApp. DApp is designed to store or access user information and user vehicle data in smart contracts, and upload some of the data to the IPFS to store the returned hash value in the smart contract. The detailed structure of the smart contract is presented in Section 4.1.1.

4.1.1. Smart Contract Structure

The proposed system handles user information, file list information uploaded by the user, and file data information, which consists of a total of four types: AllUserData, UserData, FileList, and FileHash contracts.

1. AllUser Data: A smart contract that creates UserData contracts for all users and stores the information.
2. User Data: A smart contract that stores user and user vehicle information and creates a FileList contract to store the information.
3. FileList: A smart contract that creates a FileHash contract and stores its information.
4. FileHash: A smart contract that stores the IPFS hash value of a file.

A configuration in which one smart contract owns a sub-smart contract can manage all data in one contract and increase the security of the data. In addition, it has the advantage that users can own their own smart contracts and distribute their data.

4.2. Operating Process of the System

The main implementation process of the system proposed in this study was classified into four steps: (1) registering vehicles and users; (2) storing data; (3) accessing data; and (4) sharing data.

4.2.1. Registering Vehicles and Users

Figure 2 shows the process of registering vehicle and user.

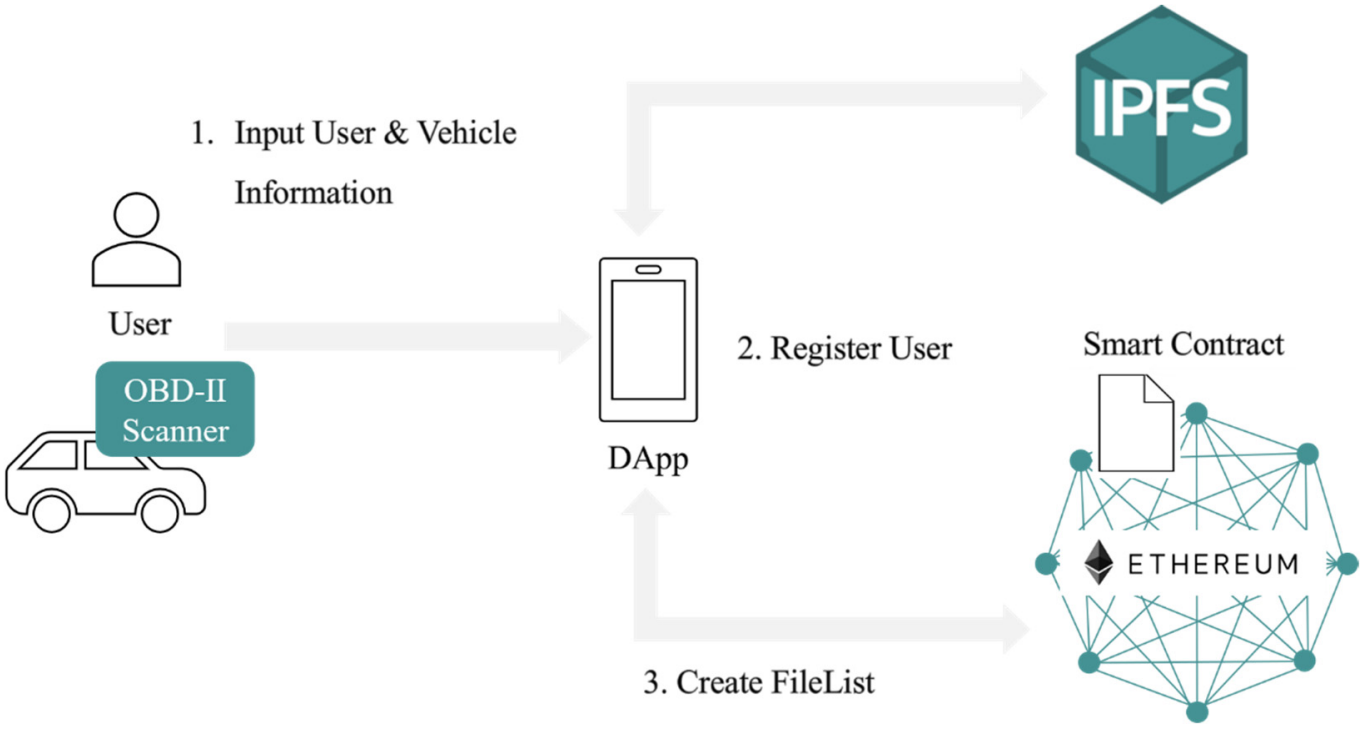

Figure 2. Process of registering vehicles and users.

Users can register to the system through a subscription, and when signing up, the user's Ethereum address, public key, file list name, and vehicle identification number (VIN) are required. The user's Ethereum address, public key, and private key are automatically entered through key extraction by accessing the keystore file in which the user's Ethereum key information is stored, and the user must enter the file list name and VIN value. When signing up, the AllUserData contract is accessed, a new UserData contract is created, and the input information is mapped and saved in a map structure. The typical information

mapped and stored is the user's Ethereum address and the vehicle's VIN value. After registering the user information, a new FileList contract is created based on the file list name entered by the user, and the address and information of the FileList contract are mapped and stored in the user's data contract.

4.2.2. Storing Data

Figure 3 shows the process of storing data.

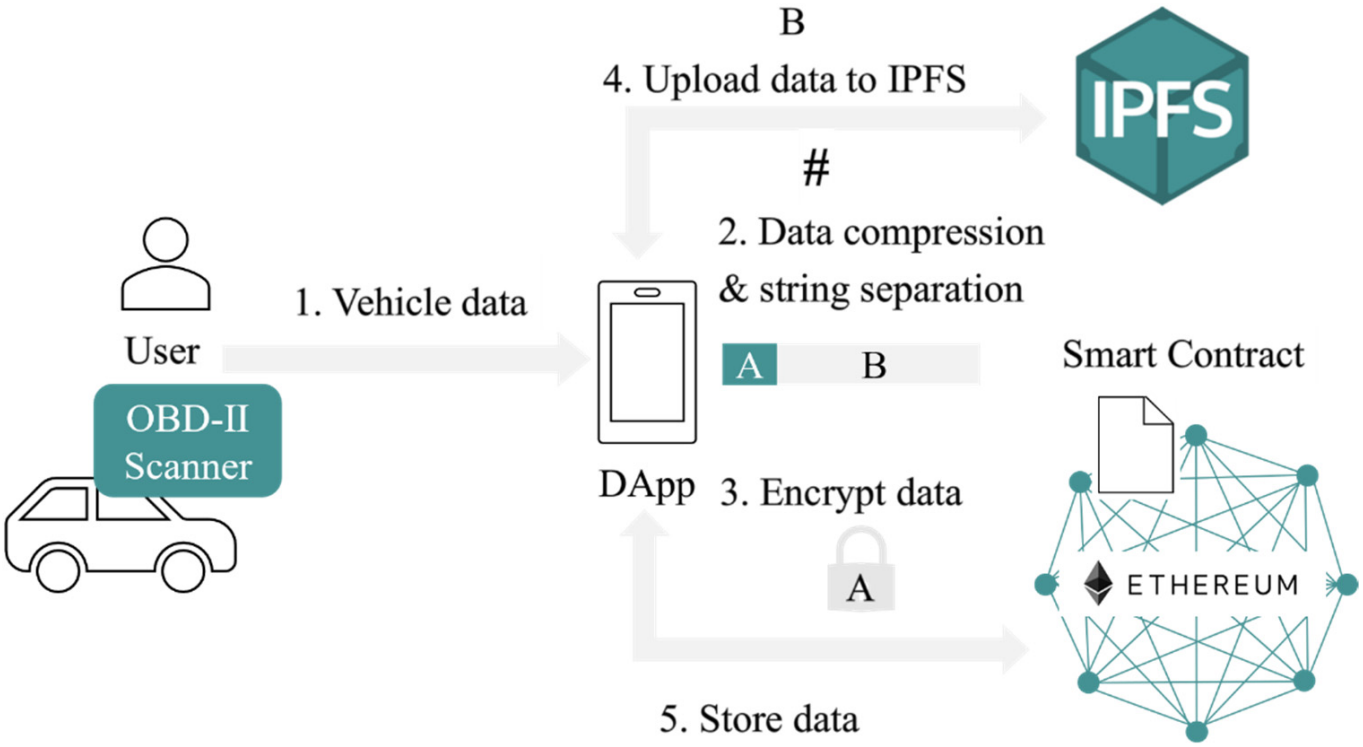

Figure 3. Process of storing data.

The process of storing vehicle data consists of a flow of receiving vehicle data and storing them in the user's contract through data compression, string separation, IPFS upload, and data encryption. The vehicle data generated in real time by an OBD-II scanner. Will be transferred to the DApp and uploaded. If the DApp is not running or the user is not logged in, the data are in the queue. When the DApp is running and the user logs in, the most recent data are from the queue are uploaded. DApp compresses the imported vehicle data to create a compressed string, separates part of the generated string, and encrypts it with the user's public key. Except for some separated strings, the rest are sent to the IPFS, and the IPFS hash value is obtained from the IPFS. The encrypted string and IPFS hash values are stored in the FileHash contract. To accomplish this, a FileList contract must be created through the UserData contract in advance. A new FileHash contract can be created through the created FileList contract, and the encrypted string and IPFS hash values are stored in the FileHash contract. At this time, contract information, including the address of the FileHash contract, is managed and stored in the FileList contract. In addition, when creating a FileList contract, the user's Ethereum address is mapped and stored in the contract to indicate that the user has access to the contract.

4.2.3. Accessing Data

Figure 4 shows the process of accessing data.

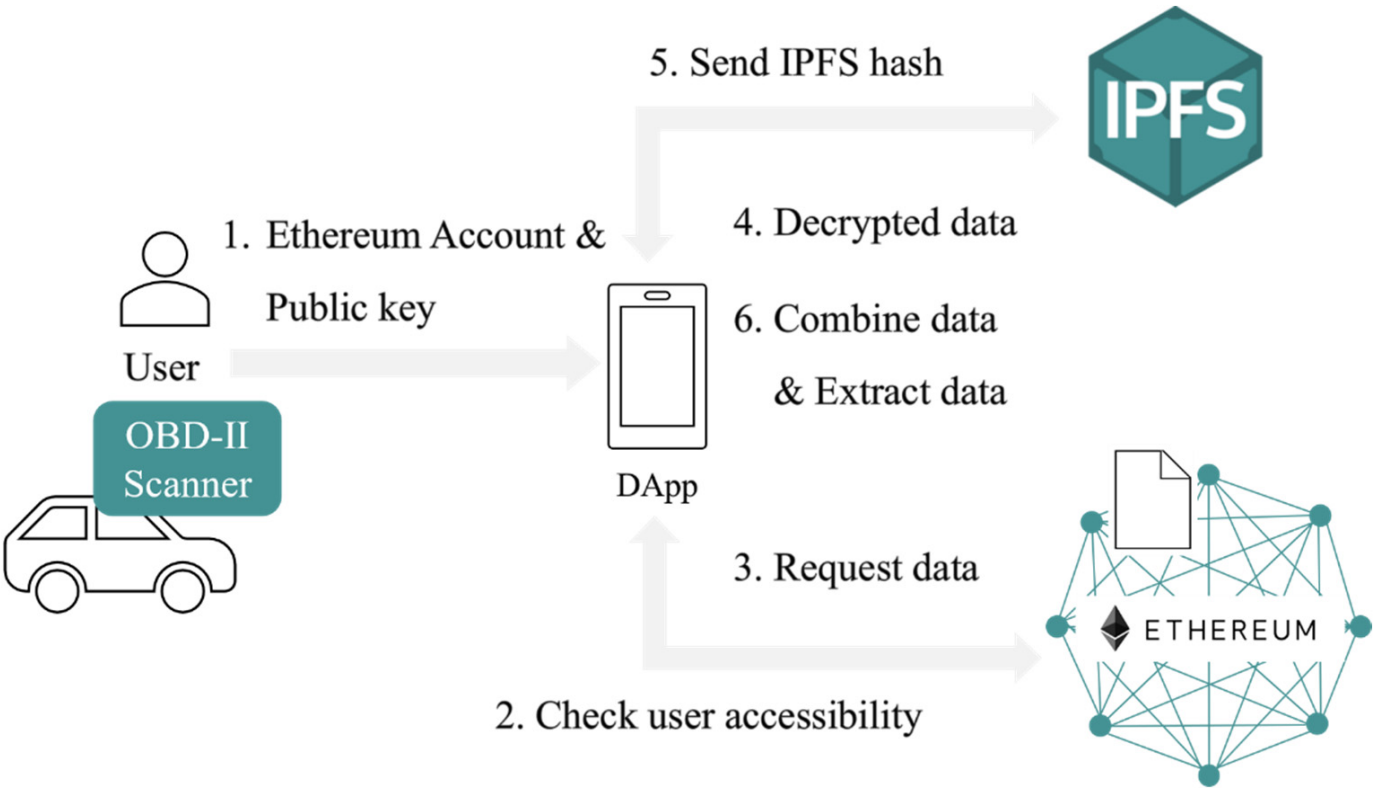

Figure 4. Process of accessing data.

To access the desired data, the user can enter the user's Ethereum address, private key, and FileList contract name and file name into the DApp and begin to access the FileList contract stored in the user's UserData contract. To access the FileList contract, user authentication is performed by checking the mapping information to determine whether they have access rights to the FileList, and if it they have, users can obtain the address of the matching FileHash contract based on the file name of the desired file. Users can then access the contract with the obtained address to access the encrypted string and IPFS hash value. The encrypted string can be decrypted with the user's private key, and the IPFS hash value can be transmitted to the IPFS to obtain the data. Then, the DApp combines the decrypted string with the string returned from IPFS, decompresses it to obtain data, and finally, the user can check the vehicle data.

4.2.4. Sharing Data

Figure 5 shows the process of sharing data.

The process of sharing data largely consists of creating a FileList contract to be shared. The data requester provides the user with his Ethereum address and public key, and the user provides their information, the information received from the data requester, and the name of the new FileList contract to be shared with the data requester to the DApp. The DApp accesses the user's data contract based on the input information to create a new shared FileList contract, and the information of the newly created shared FileList contract is stored in the user data contract and insurance company. In addition, the user's Ethereum address and the data requester's Ethereum address are mapped and stored in the shared FileList contract to specify their permission to access the FileList contract. After creating the shared FileList contract, the user provides the DApp with the user's Ethereum address, private key, the data requester's Ethereum address and public key, the new FileList contract name, and the data file name to be shared. To obtain data to share, DApp accesses the user's FileList contract and obtains the address of the FileHash contract that matches the file name provided by the user. Then, it retrieves the contract through the

address of the acquired FileHash contract to obtain the encrypted string, IPFS hash value, and decode the encrypted string using the user's private key. To share the user's data with the data requester, the decrypted string is encrypted with the data requester's public key, and the IPFS hash value is obtained by creating a new FileHash contract through the shared FileList contract, and the encrypted data are stored in it. The address of the FileHash contract is stored in the shared FileList contract, and the user shares the newly created shared FileList information, and the file name is shared with the data requester. The data requester can check the contents of the file through the same process as the user's data access process; when decrypting the file, the data can be verified by decrypting the file with the data requester's own private key rather than the user's private key. In addition, to remove the data requester's file access rights, the user who is the owner of the data must enter the FileList contract name and the data requester's Ethereum address. Then, by accessing the user's FileList contract in the DApp and changing the data requester's mapping information stored in the user's FileList contract, the user can remove the data requester's access to their data.

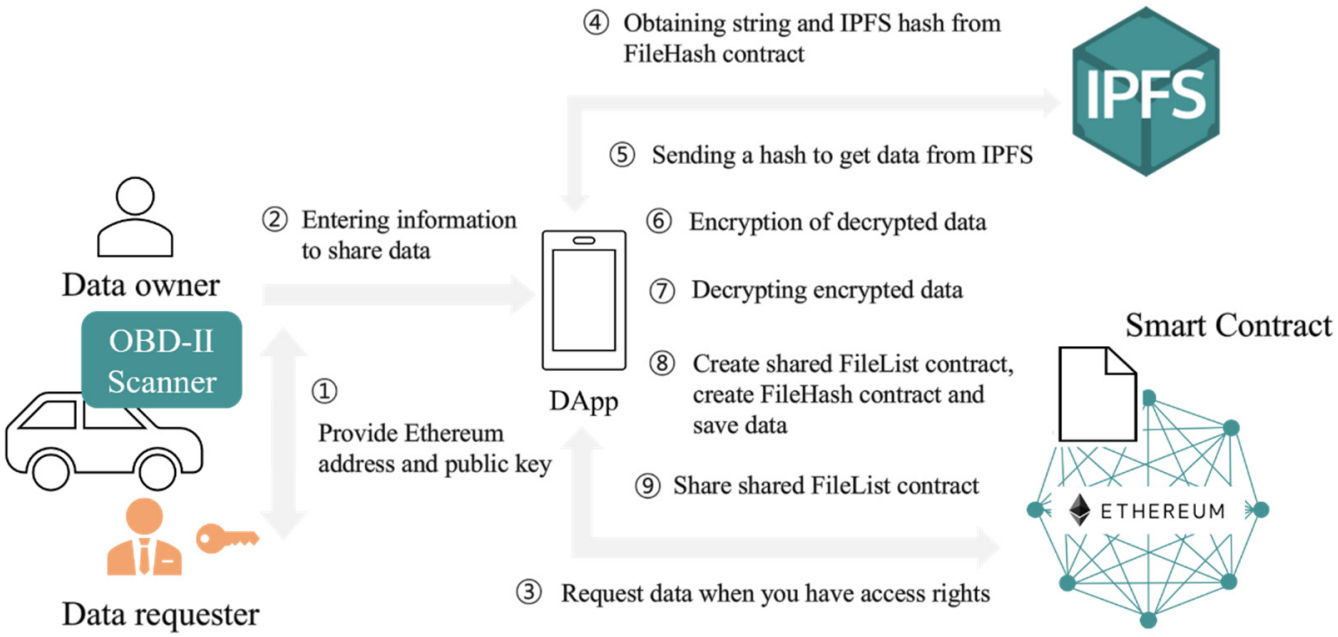

Figure 5. Process of sharing data.

5. Implementation

To develop a DApp in a local environment more easily, a blockchain network was built using Ganache [14], an Ethereum RPC client that can be installed and used locally, and Truffle [15] is used as a framework for compiling and distributing smart contracts. DApp was developed based on a web browser and a web server was built with Node.js. The DApp accesses the smart contract by communicating with the Ethereum network via JSON RPC through Web3.js [16], an Ethereum JavaScript API. In addition, the web server communicates with the IPFS network to upload data, and to access IPFS, go-ipfs [17], which can run the IPFS network in the local environment, was used. For the layout and basic implementation of a web browser-based DApp, we refer to the free open source [18] that deals with the web browser-based DApp development tutorial using IPFS, and we implemented the system's four main processes to complete the system.

Web-Based DApp

The web browser-based DApp page implemented based on the design is shown in Figures 6–9.

The login page in Figure 6 requests the user's Ethereum address and public key from the web server to load the value, and the VIN value and FileList contract name can be entered by the user. The sign-up page has a similar layout, and when the user clicks the sign-up button after entering all values, the web server creates a new smart contract based on the entered values and stores the information. In addition, when login and registration are successful, a data upload request is executed, and the web server fetches the vehicle data and performs the data uploading.

Figure 6. Login page.

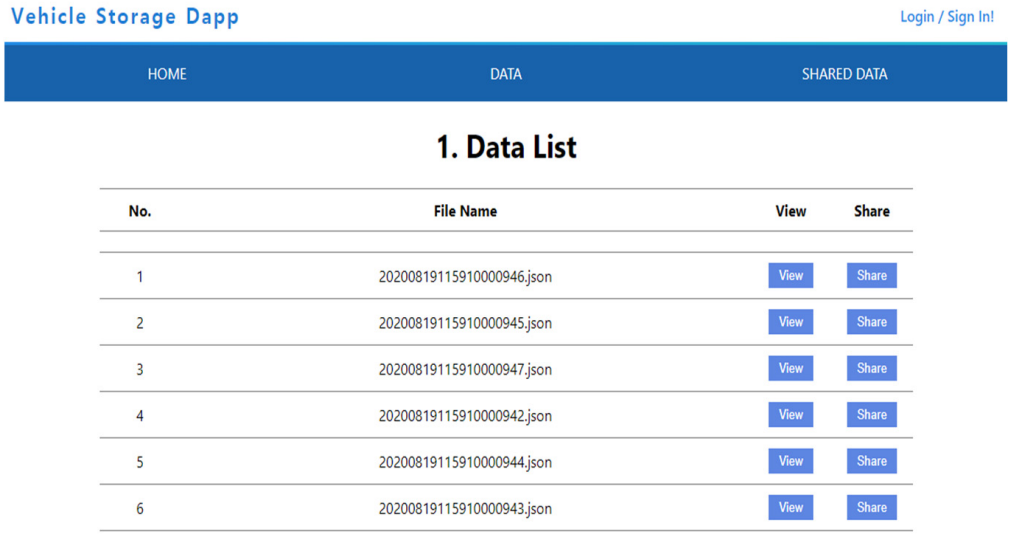

Figure 7. Data page.

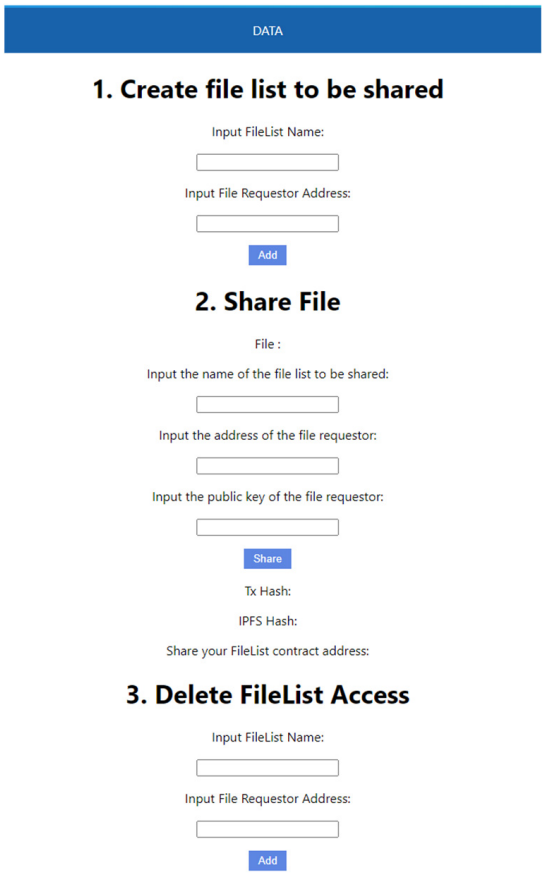

Figure 8. Share page.

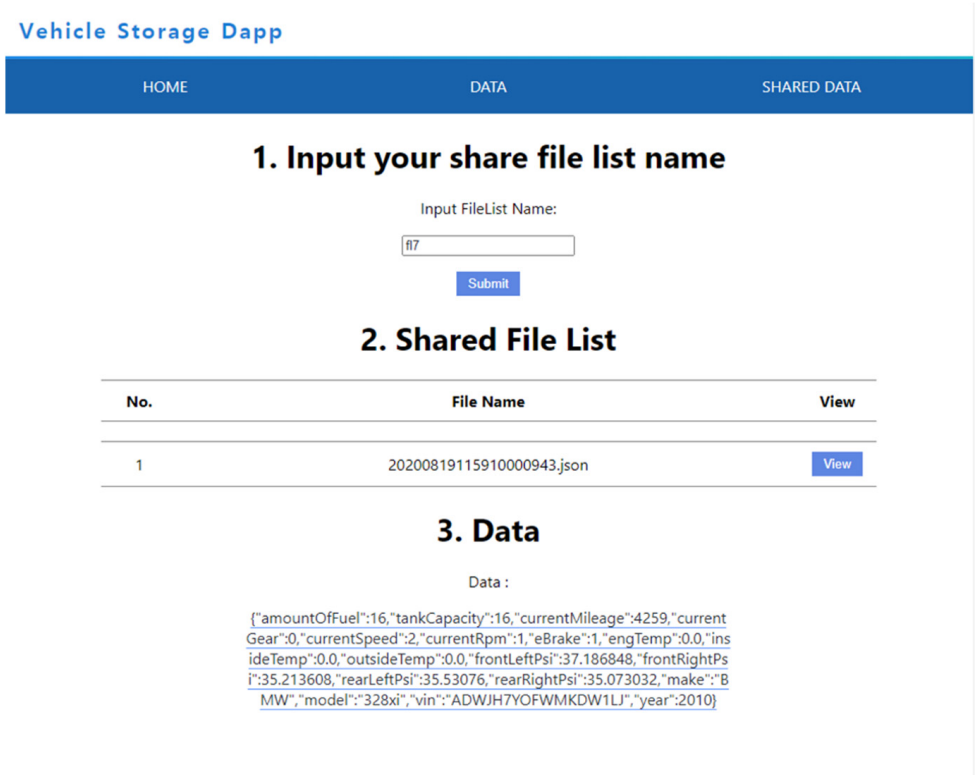

Figure 9. Share Data page.

In the data page of Figure 7, the user can check the list of file data stored in his FileList contract, and when clicking the View button, the data access process is requested, and the data can be viewed. When the user clicks the share button, they are taken to the sharing page.

Figure 8 shows the sharing page. The sharing process consists of: (1) creating a shared FileList; (2) sharing files; and (3) removing access rights. The user can request data sharing by entering the user's information, the information of the data requester who will share the data, and the file list name.

The Share Data page, shown in Figure 9, is where users can check their shared file list. The Share Data page consists of: (1) enter the shared file name; (2) list of shared files; and (3) check data. When the user enters the shared file list name, the user's file data list is displayed, and the user can click the View button to check the data.

6. Scenario

This section describes the most representative scenarios to understand the data sharing process, which is one of the key processes of the system. In this scenario, it is assumed that the insurance company provides services such as discounting money through the analysis of vehicle data of the insured user. The user must be able to share their vehicle data with the insurance company to use the service. Insurance companies request vehicle data from users and provide their Ethereum addresses and public keys. The user selects the vehicle data to share from their vehicle data in the DApp, and then requests the creation of a new FileList contract to be shared with the insurance company. After creating a shared FileList contract through DApp, the user enters their own Ethereum address and private key, the Ethereum address and public key of the insurance company, the FileList name to be shared with the insurance company, and the vehicle data file name, and requests data sharing. According to the data sharing request, the DApp finds the corresponding FileHash contract in the user's FileList contract through the received vehicle data file name and obtains the encrypted string and IPFS hash value. The encrypted string is decrypted with the user's private key and then encrypted again with the public key of the insurance company. The IPFS hash value obtained from the user's FileHash contract and the string encrypted with the insurance company's public key are stored in the newly created FileHash contract through the shared FileList contract created in advance. Through this process, users can share vehicle data with insurance companies. Insurance companies that have shared vehicle data can check the shared data list by entering the shared FileList name on the 'Share Data' page to read the user's vehicle data and click the 'View' button to check the contents of the data. At this point, the DApp obtains the encrypted string and IPFS hash value through the same process as the user's data access process, decrypts the encrypted string with the insurance company's private key, and provides the IPFS hash value to the IPFS to obtain the compressed string. Thereafter, the string that combines the returned compressed string and the decoded string is decompressed, and the result is provided to the insurance company through the DApp, so that the insurance company can check the data shared by the user. In addition, if the user wants to remove the insurance company's data access rights, they can do so by entering the shared FileList contract name and the insurance company's Ethereum address into the DApp.

7. Experiments

This section describes the results of the experiments conducted to evaluate the system performance. To test the performance of the proposed system, an experiment was conducted using an open source OBD data generator, carOBDDataGen [19], that generates vehicle data. In addition, the function of DApp fetching and uploading the latest data from OBD-II scanners is designed; however, in this experiment, a local folder is used as a queue. The experimental environment specifications were as follows: CPU: Intel Core i3-5005U, RAM: 8GB, GPU: Intel HD Graphics 5500, and Windows operating system. The version of

Node.js used as a web server was 11.4.0, and Truffle 5.1.15, Ganache 2.1.2, go-ipfs version 0.6.0 was used. We used 802.11n WiFi for networking method of the DApp.

7.1. Data Processing Speed

The first experiment involved the vehicle data processing speed. First, we measured and analyzed the speed of uploading the data processed by the proposed system to the smart contract and the uploading speed of the original data that did not go through data processing through the proposed system to the smart contract. The experiment was conducted using 100 vehicle OBD data. The original size of the data used in the experiment was approximately 328–339 bytes, and the experimental results are shown in Figure 10a.

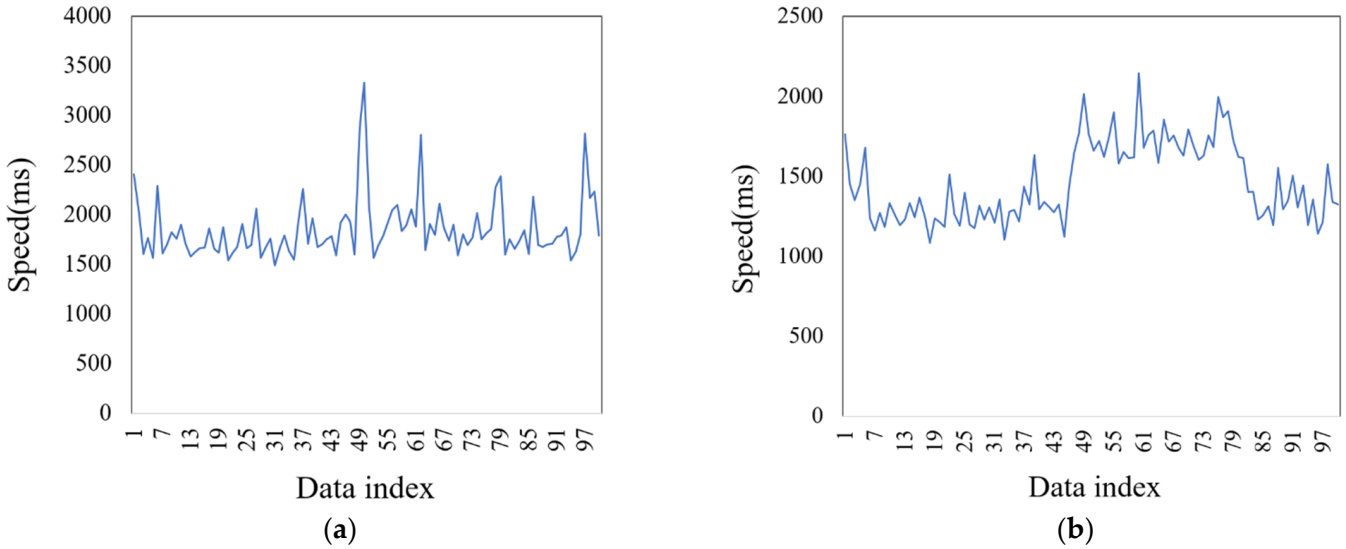

Figure 10. (**a**) Data upload speed through the system, (**b**) original data upload speed.

As a result of measuring the upload speed of a total of 100 vehicle OBD data, it took approximately 1500–3500 ms, and approximately 1854 ms on average. To prove that the data upload speed through the proposed system is excellent, we also measured the speed of uploading the original data and compared the performance. The results are shown in Figure 11b.

Uploading the original data took an average of 1451 ms, which was faster uploading the data through the proposed system. This is because the data processed by the proposed system was 358 bytes, but the original data was approximately 328 to 339 bytes. With the increase in the size of the original data, the time it takes to upload the original data to the smart contract will increase proportionally. On the other hand, uploading data through the proposed system does not change the rate of uploading data to smart contracts because the size of the data being uploaded to smart contracts does not change. To prove this, we compared the upload speed by conducting an experiment with 100 data points, which is 10 times the data size used in the previous experiment.

As a result of the measurement, the uploading time of the original data with the size of the data 10 times larger was approximately 2617 ms; thus the speed increased significantly compared to the previous experiment. On the other hand, the uploading speed of data through the proposed system took approximately 1852 ms on average, similar to before, even when the data size increased significantly.

The last experiment we conducted regarding the data processing speed was the measurement of the bandwidth of the vehicle data upload process, and we tested whether the vehicle data can be processed in real time. The average time taken to upload 100 data of an average size of 337 bytes was 114,887 ms, meaning it took approximately 114,887 ms

to upload approximately 33,700 bytes of data, and the system output performance was approximately 2346 bps.

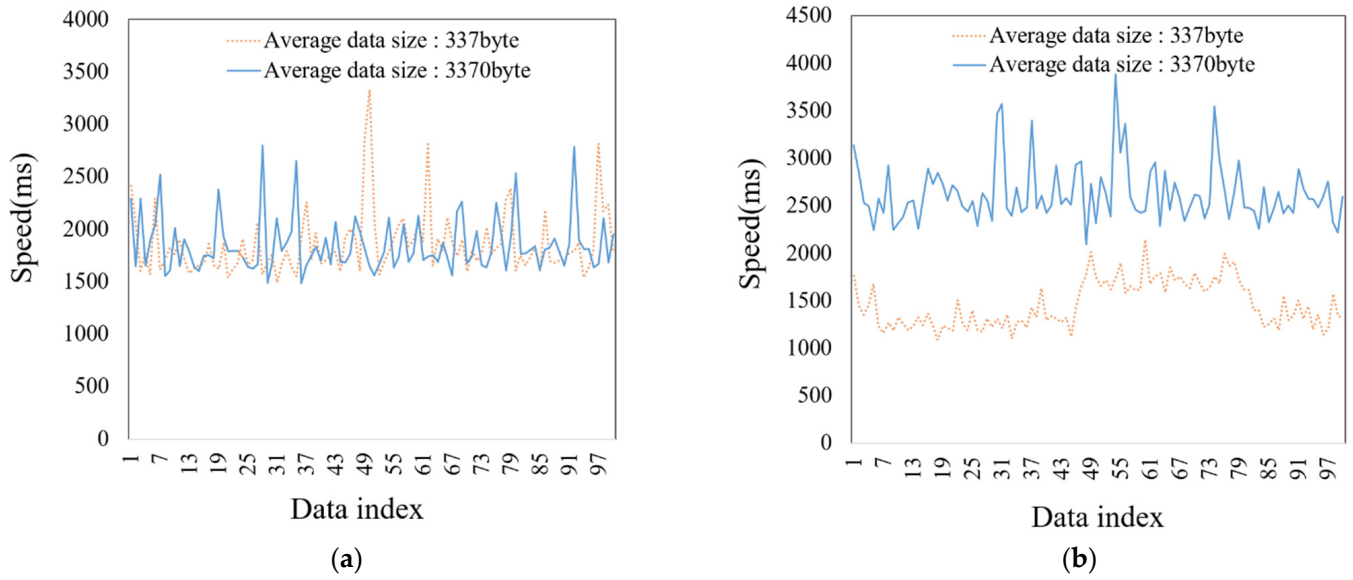

Figure 11. (**a**) Data upload speed through the system for each data size, (**b**) original data upload speed for each data size.

7.2. Fuel Consumption When Uploading Data

To execute the Ethereum smart contract, fuel must be paid. Therefore, the fuel consumption when the original data was uploaded was measured and compared with the fuel consumption when the data were uploaded through the proposed system. The unit of fuel consumption is Gwei, the smallest unit of Ether, the Ethereum cryptocurrency. 10^9 wei is called 'Gwei'.

Table 1 shows that when uploading original data with an average size of 337 bytes, fuel consumption was on average about 640,491 Gwei, and when uploading data of size 3370 bytes, fuel consumption on average is approximately 2,763,853 Gwei. This shows that the fuel consumption increases as the data size increases. On the other hand, when data were uploaded through the proposed system, there was no significant difference in fuel consumption, even if the data size increased. Based on the experimental results shown in Table 1, the fuel consumption when uploading data with an average data size of 337 bytes was measured as 662,583 Gwei on average, and when uploading data of 3370 bytes, fuel consumption was measured as 662,775 Gwei on average. This experiment confirmed that uploading vehicle data through the proposed system shows similar fuel performance as with the original system even when the size of vehicle data increases.

Table 1. Fuel usage comparison by upload method.

Upload Method	Average Data Size	
	337 Bytes	3370 Bytes
Upload original data	640,491	2,763,853
Upload through the system	662,583	662,775

(Unit: Fuel usage, Gwei).

7.3. Processing Speed by Section

The next experimental item is the processing speed by process. The main processes that need to be taken to upload vehicle data can be classified into data encryption, data upload to IPFS, and data upload to smart contracts. The performed experiment aimed to measure the rate of data processing and which section required the most time. The

experiment was carried out using 100 vehicle OBD data, as in the previous experiment, and the original size of the data was approximately 328 to 339 bytes. The results of the experiments are presented in Table 2.

Table 2. Data upload process speed by section.

Section	Speed (ms)
Save data to smart contract	1183
Upload data to IPFS	59
Data encryption	11

7.4. Shared Processing Speed

The last item tested was the amount of time it took for users to share data with others. Data sharing time measures the total time spent sharing data, recalling shared data, and reading data. The experiment used 100 data with an average data size of 337 bytes, and the results are shown in Figure 12.

Figure 12. Data sharing speed.

As a result of the measurement, it took an average of 3370 ms, it took a lot of time to upload shared data, and the data reading speed was high.

Through the above experiments, we were able to confirm the processing performance of the system and confirmed that the section that most affects the vehicle data upload speed of the system is the upload section of the smart contract. In addition, research to improve the time to upload data to smart contracts will be needed in the future to reduce upload speed.

8. Conclusions

This paper proposes a system that safely and efficiently stores vehicle data using Ethereum blockchain and IPFS and develops smart contracts and Ethereum DApp to make it convenient for users to store and manage their vehicle data. The proposed system encrypts vehicle data and stores the hash value of vehicle data in a blockchain using a digital signature method to ensure the security and integrity of the data, and efficiently store and manage vehicle data with large data sizes. To measure the performance of uploading vehicle data, we measured and compared the speed of uploading data without

using the system and using the system. The experiment was conducted using data of various sizes, and it was confirmed that the larger the data size, the faster and more efficient the uploading was. In addition, in terms of fuel usage, it was confirmed that the upload method using the system consumes a similar amount of fuel even when the data size increases. The part that most affected the data upload speed was the uploading of data to the smart contract; if this was improved, the data upload speed through the system could be further improved.

In addition, by using the proposed system, vehicle data corresponding to personal information can be safely and efficiently stored and accessed, and by sharing vehicle data with other users through public key encryption methods and smart contracts, users take control of their own data. The system is particularly suitable for use with the purpose of sharing data with insurance companies. In the future, the security of the simple user authentication process will be strengthened and DApp implemented based on web browsers and on Android apps. In addition, we could implement faster data uploading by improving the process of uploading data to the smart contract, which is the most time-consuming process.

Author Contributions: Conceptualization, H.Y. and S.P.; methodology, H.Y., S.P.; software, H.Y.; validation, H.Y., S.P.; formal analysis, H.Y., S.P.; investigation, H.Y.; resources, H.Y.; data curation, H.Y.; writing—original draft preparation, H.Y.; writing—review and editing, H.Y., S.P.; visualization, H.Y.; supervision, S.P.; project administration, S.P.; funding acquisition, S.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Bisa Research Grant of Keimyung University in 2020.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. 40+ Corporations Working on Autonomous Vehicles. Available online: https://www.cbinsights.com/research/autonomous-driverless-vehicles-corporations-list/ (accessed on 4 January 2021).
2. What Does OBD Stand for? Available online: https://www.noregon.com/what-is-obd/ (accessed on 4 January 2021).
3. Crosby, M.; Pattanayak, P.; Verma, S.; Kalyanaraman, V. BlockChain Technology: Beyond Bitcoin. *Appl. Innov.* **2016**, *2*, 6–10.
4. Archer-Soft. What Is the Best OBD2 Android/IOS Apps for Your Car? Available online: https://archer-soft.com/blog/what-best-obd2-androidios-apps-your-car (accessed on 4 January 2021).
5. Haig, S. Bitcoin Block Size, Explained. Available online: https://cointelegraph.com/explained/bitcoin-block-size-explained (accessed on 4 January 2021).
6. Krzanich, B. Data Is the New Oil in the Future of Automated Driving. Available online: https://newsroom.intel.com/editorials/krzanich-the-future-of-automated-driving/#gs.5wt1jq (accessed on 4 January 2021).
7. Benet, J. IPFS—Content Addressed, Versioned, P2P File System. *arXiv* **2014**, arXiv:1407.3561.
8. Buterin, V. A next-generation smart contract and decentralized application platform. *White Pap.* **2014**, *3*, 1.
9. Ethereum Is a Global, Open-Source Platform for Decentralized Applications. Available online: https://ethereum.org/en/what-is-ethereum/ (accessed on 4 January 2021).
10. Cai, W.; Wang, Z.; Ernst, J.B.; Hong, Z.; Feng, C.; Leung, V.C. Decentralized applications: The blockchain-empowered software system. *IEEE Access* **2018**, *6*, 53019–53033. [CrossRef]
11. Khatal, S.; Rane, J.; Patel, D.; Patel, P.; Busnel, Y. FileShare: A Blockchain and IPFS framework for Secure File Sharing and Data Provenance. In Proceedings of the International Conference on Modelling, Simulation & Intelligent Computing (MoSICom 2020), Dubai, United Arab Emirates, 29–31 January 2020; pp. 2–7.
12. Steichen, M.; Fiz, B.; Norvill, R.; Shbair, W.; State, R. Blockchain-based, decentralized access control for IPFS. In Proceedings of the 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Halifax, NS, Canada, 30 July–3 August 2018; pp. 1499–1506.
13. Liang, X.; Zhao, J.; Shetty, S.; Liu, J.; Li, D. Integrating blockchain for data sharing and collaboration in mobile healthcare applications. In Proceedings of the 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Montreal, QC, Canada, 8–13 October 2017; pp. 1–5.
14. GANACHE OVERVIEW. Available online: https://www.trufflesuite.com/docs/ganache/overview (accessed on 4 January 2021).
15. Truffle Overview. Available online: https://www.trufflesuite.com/docs/truffle/overview (accessed on 4 January 2021).
16. web3.js. web3.js—Ethereum JavaScript API. Available online: https://web3js.readthedocs.io/en/v1.2.6/ (accessed on 4 January 2021).
17. IPFS Distributions. Available online: https://dist.ipfs.io/#go-ipfs (accessed on 4 January 2021).

18. Alex6614, IPFS-Ethereum-Tutorial. Available online: https://github.com/Alex6614/IPFS-Ethereum-Tutorial (accessed on 4 January 2021).
19. Jpatel-Pivotal, carOBDDataGen. Available online: https://github.com/jpatel-pivotal/carOBDDataGen (accessed on 4 January 2021).