SC^{2}Share: Smart Contract for Secure Car Sharing

Akash Madhusudan[1], Iraklis Symeonidis[1,2], Mustafa A. Mustafa[1,3], Ren Zhang[1,4] and Bart Preneel[1]
[1]*imec-COSIC, KU Leuven, Belgium*
[2]*SnT-APSIA, University of Luxembourg, Luxembourg*
[3]*School of Computer Science, University of Manchester, U.K.*
[4]*Nervos, Beijing, China*

Keywords: Smart Contracts, Ethereum, Car Booking and Payments, Security, Privacy, Car Sharing.

Abstract: This paper presents an efficient solution for the booking and payments functionality of a car sharing system that allows individuals to share their personal, underused cars in a completely decentralized manner, annulling the need of an intermediary. Our solution, named SC^{2}Share, leverages smart contracts and uses them to carry out secure and private car booking and payments. Our experiments on SC^{2}Share on the Ethereum testnet guarantee high security and privacy to its users and confirm that our system is cost-efficient and ready for practical use.

1 INTRODUCTION

Recent years have witnessed an exponential growth of the sharing economy (Cusumano, 2017). It enables individuals to share their personal assets for financial gain, helping others who temporarily need these assets. As of 2018, all practical solutions supporting the sharing economy rely on centralized platforms for exchanging information and settling payments. Unfortunately, a series of incidents indicates that these platforms are not as trustworthy as they promise to be. First, their centralized membership system and pricing scheme allow them to undermine *fairness*. Airbnb, an online marketplace for sharing accommodation, has been in the news for targeting a subgroup of the population to exclude them from using its services (Collins, 2018). This highlights the fact that there is no mechanism to prevent them from misusing this power, for instance, to target a minority. Furthermore, the peer-to-peer car sharing platform Uber charges its customers based on an algorithmic prediction on how much they are willing to pay, rather than the services they receive (Newcomer, 2017). Second, by keeping all data in one database, these platforms become attractive targets for attackers, resulting in several data breaches that violate the *privacy* of millions of users (Carsten, 2016; Lee, 2017). Along with the fairness and privacy constraints, using these platforms entails an added cost to both, the user and the service provider. (Thellmann, 2018).

Fully decentralized solutions based on blockchain (Sharma, 2018) are seen as a solution to the fairness issues of these centralized platforms. A blockchain is a trustless platform that allows any user to transfer funds and execute *smart contracts*– decentralized applications. Although the open nature of the blockchain provides better fairness guarantees, it is unclear how these solutions protect the users' privacy. Of the two blockchain-based asset-sharing schemes the authors are aware of, La'Zooz stores the private information in a centralized database (La'Zooz, 2015), while the relevant code of Slock.it is closed-source, possibly under development (Jentzsch, 2016). Moreover, these systems provide a limited set of functionalities in comparison with their centralized competitors. La'Zooz only supports Consumer-to-Consumer (C2C) car-pooling in their advertisement, therefore it is unclear whether they can support C2C personal car sharing. As it appears, users of Slock.it cannot cancel a request once it is registered, therefore any misoperation might lead to financial loss for the user. Moreover, Slock.it has not yet been extended to car sharing. Robust car access provision protocols, such as SePCAR (Symeonidis et al., 2017), are also limited in terms of functionality, as they do not provide a solution to booking and payments.

This work proposes a novel peer-to-peer car

Madhusudan, A., Symeonidis, I., Mustafa, M., Zhang, R. and Preneel, B.
SC2Share: Smart Contract for Secure Car Sharing.
DOI: 10.5220/0007703601630171
In *Proceedings of the 5th International Conference on Information Systems Security and Privacy (ICISSP 2019)*, pages 163-171
ISBN: 978-989-758-359-9

booking and payments system, named *SC²Share*. SC²Share works along the existing car access provision protocols such as SePCAR and uses a smart contract to register car sharing offers, match requests and settle payments. Compared with existing schemes, our system has the following advantages:

- **Fair.** Our system provides better fairness guarantees as it does not suffer from centralized price manipulation and membership exclusion.
- **Resistance against Data Breach.** There is no central point of failure in our design. Most interactions happen in a peer-to-peer fashion between the car owner and the consumer, which involves no intermediary. Sensitive information stored on the blockchain is encrypted with the receiver's public key, which is different for each participant. A compromised private key will not affect other users of the system.
- **Complete Functionalities.** Our system offers complete functionalities that cover all typical execution paths in an Uber car sharing instance, including several conflict handling capabilities that ensure financial safety of each involved party against an adversary, the other party, and misoperation by the user himself. To the best of our knowledge, these combination of features is not offered in any existing decentralized car sharing platform.
- **Cost-effective.** As an additional advantage, our system is more cost effective to use due to the absence of a commission. The only cost involved is the deployment and transaction costs incurred by the blockchain. The experiments with our deployed contract in Ethereum testnet highlights the operational costs and efficiency of SC²Share.

The remainder of this paper is organized as follows. Section 2 presents the necessary background. Section 3 presents the system model, threat model and design requirements used in our design. Section 4 describes SC²Share, followed by its evaluation in Section 5. Finally, Section 6 concludes this paper and it offers future research directions.

2 BACKGROUND

Since SC²Share handles only the booking and payments aspect of car sharing, we first give a brief overview of SePCAR, a secure and privacy-enhancing car access provision protocol to generate and revoke car access tokens (Symeonidis et al., 2017). Then, we briefly introduce smart contracts in Ethereum - the main building block of SC²Share.

2.1 SePCAR

SePCAR (Symeonidis et al., 2017) is a car sharing scheme that offers strong security and privacy properties; it extends earlier work described in (Symeonidis et al., 2016). The system consists of various functional components. We only list the ones that are relevant to our system. An *Owner* is a user who is willing to share his car, a *consumer* is the user who wants to rent a car and *authorities* are the entities responsible for ensuring that the entire system is legal as well as for resolving any disputes between users.

The SePCAR protocol starts with the mutual agreement of booking details by the owner and consumer. Due to the presence of sensitive information such as the identities of the owner, consumer and the car along with its usage duration and location, the system server encrypts these booking details and an access token is generated. This access token is then stored in a public ledger, where it is retrieved by the consumer. Subsequently, a consumer uses this token to access the car without revealing his private information. SePCAR also guarantees the confidentiality of the car key and booking details.

2.2 Smart Contracts in Ethereum

Ethereum is a cryptocurrency with the third largest market capitalization (CoinMarketCap, 2018). Aiming at realizing a "world computer", Ethereum allows users to program smart contracts with a Turing-complete language and guarantees the correct execution of these contracts and the integrity of the system with its underlying blockchain (Buterin, 2014).

Smart contracts are special accounts on Ethereum blockchain, that contain code and persistant storage along with an address and balance like normal accounts (Luu et al., 2017). They are computerized protocols, that, without relying on any intermediaries, satisfy contractual conditions and minimize attacks by adversaries. As in any other computer program, the code of smart contracts also manipulates variables, and it can be invoked by sending a transaction to its address along with the required payment for its execution and parameters.

Miners, entities who embed transactions into the blockchain, are compensated by *transaction fees* in ether, the native currency of Ethereum, from the transaction initiators. The transaction fee is calculated as the total amount of *gas* consumed by the transaction execution, multiplied by the *gasPrice*, while the gas-ether exchange rate is specified in the transaction. The total gas is calculated by accumulating the gas consumption of all instructions of the execution. Each

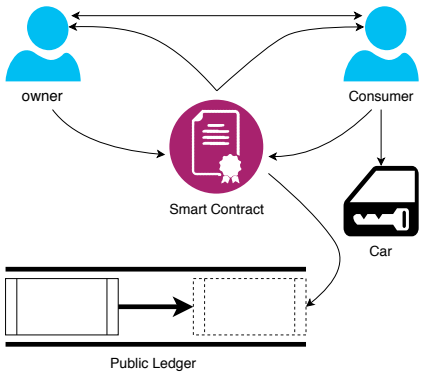

Figure 1: System model of SC^{2}Share.

instruction has a predefined gas consumption. To prevent denial of service attacks, each transaction also specifies a *startGas*. A transaction execution that exceeds that startGas cannot be carried out, however, the transaction fee still goes to the miner.

3 SYSTEM MODEL, THREAT MODEL AND DESIGN REQUIREMENTS

This section introduces the system model, the threat model and assumptions as well as the functional, security and privacy requirements used in the design of SC^{2}Share.

3.1 System Model

As illustrated in Figure 1, our system model consists of the following entities:

- *Owner* - An individual with the intention to share his underused personal car.
- *Consumer* - A person in temporary need of a car.
- *Car* - The car that is available to be shared. Access provision to this car is handled by any robust access provision protocols such as SePCAR (Symeonidis et al., 2017).
- *Public Ledger* - A structure that irrefutably records transactions and is operated by a single or several decentralized parties. All transactions made in SC^{2}Share are recorded in this public ledger.
- *Smart Contract* - This smart contract is responsible for receiving a booking request, making sure that appropriate deposits have been made by both the owner and the consumer, handling cancellation requests, dealing with fraudulent activities, and a smooth rental procedure in general.

3.2 Threat Model and Assumptions

In SC^{2}Share the following threat model is used. The owner and the consumer are not trusted to carry out all transactions honestly. The integrity and correct execution of the smart contract is guaranteed by the underlying public ledger, in our case the Ethereum blockchain. Due to the blockchain's public nature, any private information embedded in the blockchain is considered leaked.

For our system to work, we also make the following assumptions. The owner and the consumer mutually agree upon the initial booking details and are successful in keeping them confidential against any third party. We also assume that the owner and the consumer have a public/private-key pair along with their digital certificates. Lastly, the communication channels are used by the owner and the consumer are private and authentic.

3.3 Design Requirements

Our system should satisfy the following functional, security and privacy requirements.

Functional Requirements

- **Booking and Payments Process:** SC^{2}Share should handle bookings, car usage and payments in an immutable way. It should be able to receive booking requests, safeguard deposits placed by owners/consumers and handle payments.
- **Fraud Prevention:** SC^{2}Share should be able to deal with fraudulent actions taken by both owners and consumers. Example actions are not making the car availlable and not returning the car. SC^{2}Share should penalize the initiators of such fraudulent actions.
- **Cancellation:** SC^{2}Share should handle cancellations of the booking agreement and distribute the deposit according to the booking agreement.
- **Extra Time Surcharge:** In case of extra time request by the consumer, SC^{2}Share should be able to deduct the required extra amount from the consumer's deposit and transfer it to the owner.

Security and Privacy Requirements

- **Authenticity of Booking Details:** the integrity, origin and approval of the booking details should be validated by the car.
- **Non-repudiation of Access Token:** SC^{2}Share should be able to prove the authenticity of the

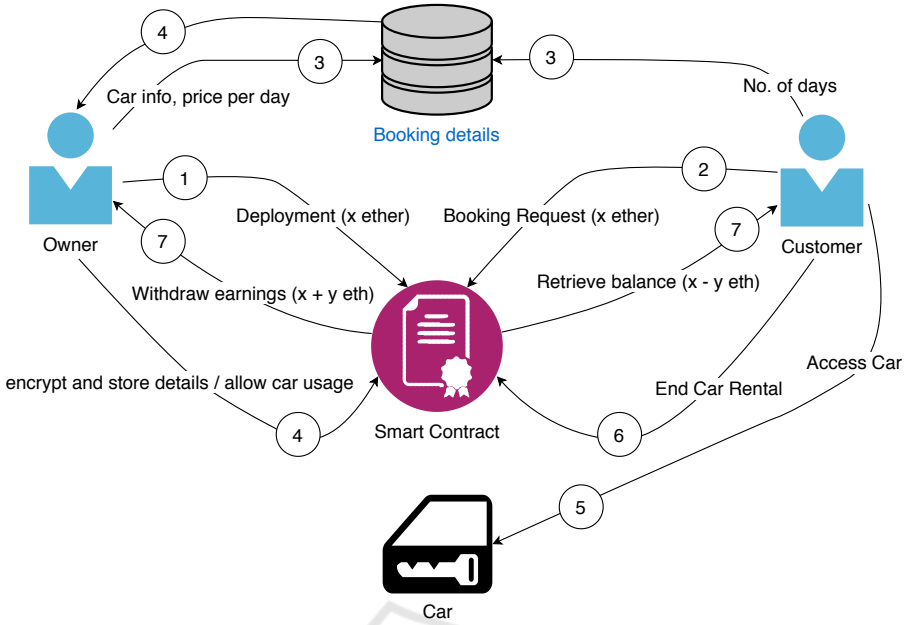

Figure 2: Overview of the SC^{2}Share system.

access token that is generated by the owner to the car and the consumer.

- **Forensic Evidence Provision**: In an adverse scenario of a fraudulent action, authorities should be able to retrieve forensic evidence from the system and gain access to specific booking details.
- **Confidentiality of Booking Details:** Only the owner and consumer should have access to the information stored in booking details. Booking details can hold personal identifiable and potentially sensitive information such as the license plate number, the location, booking time and type of the car.
- **Confidentiality of Access Token:** Only the car should have access to the information stored in the access token.
- **Anonymity of Consumer And Car:** The identity of the consumer and the car should be hidden to anyone except the owner, the consumer and the car.

4 SC^{2}Share

This section provides an overview of how SC^{2}Share handles a car sharing scenario. Then, we explain in detail the algorithms that implement the functionalities described.

4.1 Overview of SC^{2}Share

As it is illustrated in Figure 2, we provide a high-level overview of how our system works. It consists of seven steps described below.

1. An owner deploys a smart contract on the blockchain and deposits a value of x ether in it.
2. A consumer sends a request to rent the car by depositing x ether (the same amount as in step 1) in the deployed smart contract.
3. The owner and the consumer agree on mutually accepted booking details such as the required number of days to share the car, the pick-up and drop-off location and the daily price.
4. Once the details are agreed, the owner signs the booking details using his private key and encrypts them using the consumer's public key. Once encrypted, he sends these details to the storage of smart contract and allows the consumer to access the car. The encrypted details stored in the smart contract can be accessed by the consumer and decrypted using his private key. However, as seen in Figure 3, we do not encrypt all information stored in booking details.
5. The consumer accesses the car and uses it for the agreed amount of time.
6. After completing the trip, the consumer ends the car rental process by dropping off the owner's car at the agreed location.

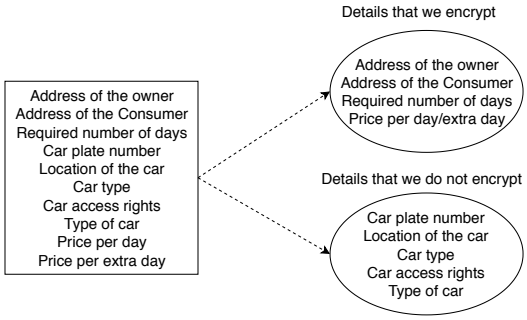

Figure 3: List of Booking Details.

7. Once the process is over, both the owner and the consumer can separately withdraw their earnings and balances, respectively.

4.2 SC^{2}Share in Detail

This section describes SC^{2}Share in detail. We first present the main protocol followed by the description of the cancellation, car access and extra time functionalities.

Algorithm 1 depicts a pseudo-code of how our system works. As explained in the overview, the system is initiated by the owner, which is followed by the consumer sending a request for booking. Once the consumer is set as the current driver of the car, the booking details are mutually agreed before being encrypted and stored on the blockchain. We use external libraries such as eth-crypto and eth-ecies to carry out the encryption off-chain, and to generate the identities of the owner and the consumer (libertylocked, 2018; pubkey, 2018). The Javascript code that executes with the smart contract, and the smart contract itself are available online.

While storing the details in the smart contract, the owner has to set allow car usage value to 'true' in the smart contract to inform SC^{2}Share that the car can be used by the consumer. Once the consumer has access to the car, he can use it for the decided time. If they exceed the agreed time limit, the extra time surcharge is handled by Algorithm 4. In order to calculate extra time, the smart contract keeps deducting the mutually agreed 'price per extra hour' from the consumer's deposit. SC^{2}Share also has a fail-safe option, that is activated if the consumer fails to return the car. It awards the owner the total deposit stored in the smart contract, although it does not yet deal with the post-theft scenario. A possible solution to the post-theft scenario is provided by insurance businesses for sharing economy industry, such as SafeShare.

In cases where either the owner or the consumer wants to cancel the booking, Algorithm 2 comes into play. The pseudo-code explains how SC^{2}Share handles the cancellation at different stages of the system. If the owner cancels the booking before allowing car usage, he is not penalized, but if they do it after allowing the car usage, then SC^{2}Share ensures that the consumer gets an incentive. The customer also pays the penalty based on whether they cancel the booking before accessing the car, or after.

If the owner fails to give car access to the consumer after they have made the deposit, Algorithm 3 depicts how the owner is penalized for committing a fraudulent action as SC^{2}Share awards the total deposit stored in the smart contract to the consumer.

Once the consumer has successfully ended the car rental (see Algorithm 1), SC^{2}Share calculates the amount that has to be deducted from their deposit and adds it to the owner's deposit, after which the owner and consumer can separately withdraw their money.

Algorithm 1: SC^{2}Share main.

```
1:  procedure SC2SHARE
2:  Owner Deploys Contract
3:      if Deployment successful then
4:          Customer Requests for Car Booking
5:          if Request successful then
6:              currentDriverAddress ← cAddress
7:              cDeposit ← x Ether
8:              oDeposit ← x Ether
9:              Mutually agree on booking details(BD)
10:             Encrypt BD and store
11:             if Owner allows car usage then
12:                 if Owner wants to cancel then
13:                     goto Algorithm 2;
14:                 if Customer access's car then
15:                     if Customer cancels then
16:                         goto Algorithm 2;
17:                     Use Car;
18:                     End Usage;
19:                     Owner withdraws earnings;
20:                     Consumer withdraws deposit;
21:                     if Extra time then
22:                         goto Algorithm 3;
23:                 else
24:                     if Customer cancels then
25:                         goto Algorithm 2;
26:             else
27:                 if Owner cancels then
28:                     goto Algorithm 2;
29:                 Customer withdraws total deposit;
30: Restart if deployment fails.
```

Algorithm 2: Cancellation functionality.

```
1:  procedure CANCELLATION
2:      currentDriverAddress ← cAddress
3:      cBalance ← consumer balance
4:      oBalance ← owner balance
5:      cDeposit ← consumer deposit
6:      oDeposit ← owner deposit
7:  Owner Allows Car Usage:
8:      if consumer access's car & cancels booking
    then
9:          x ← clientDeposit − penalty
10:         ownerBalance ← balance + x
11:         return true
12:     if owner cancels booking then
13:         x ← oDeposit − penalty
14:         cBalance ← cBalance + x
15:         return true
16:     if consumer cancels & does not access then
17:         return true
18: Owner Does Not Allow Usage:
19:     if owner cancels booking then
20:         oBalance ← oBalance + oDeposit
21:         cBalance ← cBalance + cDeposit
22:         return true
```

Algorithm 3: Car access functionality.

```
1:  procedure CAR ACCESS
2:      currentDriverAddress ← cAddress
3:      cBalance ← consumer balance
4:      if owner allows usage then
5:          carStatus ← ready
6:          consumer ← access
7:          return true
8:      if owner does not allow usage then
9:          carStatus ← notready
10:         consumer ← noAccess
11:         consumer can retrieve total deposit
12:         x ← totalDeposit
13:         cBalance ← cBalance + x
14:         return true
```

5 EVALUATION

In this section, we analyze our design and provide a description of how SC^{2}Share fulfills the functional, security and privacy requirements. We also provide the real-world deployment cost of SC^{2}Share on the Ethereum blockchain along with all the transaction costs it incurs while being used. Evaluation on the basis of total time taken is tricky as the execution time for each transaction in SC^{2}Share can only be aggregated since there is no concrete way of knowing how long a transaction would take before its included in Ethereum. As of January 2019, the median wait time between each block is ≈ 32 seconds.

Algorithm 4: Extra time functionality.

```
1:  procedure EXTRA TIME
2:      currentDriverAddress ← cAddress
3:      cBalance ← consumer balance
4:      oBalance ← owner balance
5:      cDeposit ← consumer deposit
6:      oDeposit ← owner deposit
7:  consumer Ends Car Rental:
8:      if end time =< decided time then
9:          currentDriverAddress ← null
10:         carStatus ← idle
11:         x ← cDeposit − rentCharge
12:         oBalance ← oDeposit + x
13:         cBalance ← cDeposit
14:         return true
15:     if end time > decided time then
16:         currentDriverAddress ← null
17:         carStatus ← idle
18:         x ← cDeposit − rentCharge
19:         y ← cDeposit − chargePerExtraHour
20:         z ← x + y
21:         oBalance ← oDeposit + z
22:         cBalance ← cDeposit
23:         return true
24: Consumer Runs Away With Car:
25:     x ← totalDeposit
26:     oBalance ← oBalance + x
27:     return inform insurance
```

5.1 Functionality Analysis

SC^{2}Share uses the Ethereum blockchain to implement the booking and payment functionalities for car sharing applications. All transactions related to car rental, are immutable when published on the blockchain. Once a transaction is made using SC^{2}Share, the owner or the consumer cannot refute the ownership of it. Moreover, no outsider knows the identities of the owner and consumer, and all the sensitive information in booking details is encrypted before being stored on-chain.

The Algorithm 3 and 4 demonstrates how to mitigate a malicious owner or consumer and hence prevent *fraudelent actions*. Possible fraud behaviour is countered with a penalty of losing the deposit made in the smart contract. The Algorithm 4 also enables the functionality of *extra time surcharge*.

Moreover, in SC^{2}Share we consider all possible cancellation scenarios, and a fair settlement for both the owner and consumer in Algorithm 2. Hence we guarantee the functionality of *cancellation*.

Table 1: Deployment Costs of SC^{2}Share.

Total Size (bytes)	Deployment Cost in gas	Deployment Cost in USD
8520	1,352,283 gas	$0.21623

Table 2: Costs of Executing each Transaction in SC^{2}Share.

Transaction	Cost in gas	Cost in USD
set required days (customer)	42,131 gas	$0.00674
store encrypted details in contract (owner)	63,022 gas	$0.02739
rent car (customer)	112,220 gas	$0.01795
allow use of the car (owner)	28,655 gas	$0.00459
access the car (customer)	29,038 gas	$0.00464
cancellation of booking (owner/customer)	76,342 gas	$0.0122
ending the rental procedure (owner/customer)	82,590 gas	$0.01321
triggering the distribution of earnings (owner)	32,992 gas	$0.00528
withdrawal of money (owner/consumer)	22,099 gas	$0.00353

5.2 Security and Privacy Analysis

Before publishing the booking details in the blockchain, the owner has to encrypt and sign them. SC^{2}Share verifies the owner's signature on the booking details before giving access to the consumer to verify if the owner also agrees with the conditions set for the car rental, thus providing *authenticity of booking details*.

Car access provision protocols such as SePCAR (Symeonidis et al., 2017) use the encrypted booking details to generate the access token. SC^{2}Share verifies the origin of encrypted details in order to make sure that no one but the owner has signed them. This is achieved by comparing the address of the signer to the address of the owner. This is how we ensure *non-repudiation of access token*.

Smart contracts are auditable by nature, and in case of an incident where the owner or the consumer is involved, the SC^{2}Share can be audited to reveal private information about the owner or the consumer guaranteeing the requirement of *forensic evidence provision*.

SC^{2}Share ensures the *confidentiality of booking details* by enabling only the owner and the consumer to have access to the sensitive information stored in booking details. These details are discussed offline using a secure channel (see assumptions in Section 3) and only stored in the smart contract when encrypted.

SC^{2}Share treats the encrypted booking details as the access token and stores them in its internal storage. No one except the consumer and the owner can decipher an access token thus guaranteeing *confidentiality of the access token*.

Anonymity of consumer and car is provided by design in SC^{2}Share. On the blockchain, the consumer can choose to be anonymous by using his 20-byte address to interact with the contract and not publicizing any personal details, while the identity of the car is encrypted along with the other private information by the owner before storing it on-chain.

5.3 Deployment and Usage Cost

The deployment of a smart contract and the interaction with it has a cost for the caller of the transaction, and this cost is calculated in gas. This section discusses the deployment and interaction costs associated with SC^{2}Share. It is important to mention that the price of ether per gas unit is volatile, and these results are obtained by using the price of ether per gas unit for average confirmation time as of January 2019. To evaluate the total cost of SC^{2}Share, we look at two different costs, the *deployment* and the *transaction cost*.

Deployment Cost

The main costs for deploying a smart contract can be calculated by combining the following parameters (Wood, 2014):

- The costs associated with storing the contract code (200 gas per byte);
- Additional data storage cost on the contract (20,000 gas per 256-bit word);
- Each transaction has a base cost of 21,000 gas, and 32,000 gas have to be paid for a CREATE transaction (deployment of a new contract);

Ganache was used as a provider for web3.js to perform tests on SC^{2}Share and calculate the estimated gas for deployment, as well as the execution of each transaction. The absolute deployment cost is influenced by the size of the contract and the total bytes in

its storage. Table 1 illustrates the deployment cost of SC^{2}Share in USD. It can be calculated by multiplying the total gas with the cost per gas unit in USD.

Transaction Cost

The cost for executing each transaction can be calculated as follows (Wood, 2014):

- The base cost of each transaction (21,000 gas);
- The cost for storing a 256-bit word on the smart contract (20,000 gas);
- The cost for editing data stored in the smart contract (5,000 gas);
- The cost of making a transaction call having a monetary value (9,000 gas);

Table 2 illustrates the cost of executing each transaction in SC^{2}Share. Adding up the cost of all individual transactions, the total cost of all transactions in SC^{2}Share is USD 0.095. Hence, it can be deduced that our smart contract implementation is not expensive to deploy and operate.

6 CONCLUSIONS AND FUTURE WORK

This paper presented a fully decentralized car booking and payments system known as SC^{2}Share. This system can be incorporated with car access provision protocols to provide a secure and private car sharing environment without the need of any intermediary. We have shown that SC^{2}Share provides all major functionalities that are required for a car sharing platform, and provides security and privacy by design. The total cost of deploying and using our system on the Ethereum network sums up to USD 0.312, which in comparison to the commission fee paid to large organizations is relatively cheap. Hence, we conclude that along with being functionally sound, secure and private, SC^{2}Share is also cost effective for its users.

As future work we would like to advance our system design and implementation to work with fully encrypted booking details, including the price per day, price per extra day and required number of days which are used for calculation of payments, as well as to provide formal security and privacy proofs of the advanced system.

Another potential direction could be to adapt SC^{2}Share so that it supports the use of advanced cryptographic primitives such as zero-knowledge proofs. Ethereum Whisper could also be used to possibly remove the waiting time of customer to get car access.

ACKNOWLEDGEMENTS

This work was supported in part by the Research Council KU Leuven: C16/15/058 and by the Flemish Government through FWO SBO project SNIPPET S007619N. We would also like to thank the anonymous reviewers for their constructive feedback.

REFERENCES

Buterin, V. (2014). A next-generation smart contract and decentralized application platform. URL: http://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf, last checked on 2018-12-20.

Carsten, P. (2016). Hackers attack 20 mln accounts on alibaba's taobao shopping site. URL: https://www.reuters.com/article/alibaba-cyber-idUSL3N15J1P2, last checked on 2018-12-10.

CoinMarketCap (2018). Top 100 cryptocurrencies by market capitalization. URL: https://coinmarketcap.com/, last checked on 2018-12-20.

Collins, K. (2018). A running list of websites and apps that have banned, blocked, deleted, and otherwise dropped white supremacists. URL: https://qz.com/1055141/what-websites-and-apps-have-banned-neo-nazis-and-white-supremacists/, last checked on 2018-12-07.

Cusumano, M. A. (2017). The sharing economy meets reality. *Commun. ACM*, 61(1):26–28.

Jentzsch, C. (2016). Decentralized autonomous organization to automate governance. URL: https://download.slock.it/public/DAO/WhitePaper.pdf, last checked on 2018-12-10.

La'Zooz (2015). La'zooz white paper. URL: https://www.weusecoins.com/assets/pdf/library/LaZooz%20Blockchain%20Taxi%20Whitepaper.pdf, last checked on 2018-12-10.

Lee, D. (2017). Uber concealed huge data breach. URL: https://www.bbc.com/news/technology-42075306, last checked on 2018-12-10.

libertylocked (2018). eth-ecies. URL: https://github.com/libertylocked/eth-ecies, last checked on 2018-12-20.

Luu, L., Velner, Y., Teutsch, J., and Saxena, P. (2017). Smart pool: Practical decentralized pooled mining. *In USENIX Security Symposium*.

Newcomer, E. (2017). Uber starts charging what it thinks youre willing to pay. URL: https://www.bloomberg.com/news/articles/2017-05-19/uber-s-future-may-rely-on-predicting-how-much-you-re-willing-to-pay, last checked on 2018-12-07.

pubkey (2018). eth-crypto. URL: https://github.com/pubkey/eth-crypto, last checked on 2018-12-20.

Sharma, T. K. (2018). What does blockchain means for a sharing economy? URL: https://www.blockchain-council.org/blockchain/what-does-

blockchain-mean-for-a-sharing-economy/, last checked on 2018-12-10.

Symeonidis, I., Aly, A., Mustafa, M. A., Mennink, B., Dhooghe, S., and Preneel, B. (2017). Sepcar: A secure and privacy-enhancing protocol for car access provision. *in the 22nd European Symposium on Research in Computer Security (ESORICS17), ser. LNCS, vol. 10493. Springer, 2017*, pages 475–493.

Symeonidis, I., Mustafa, M. A., and Preneel, B. (2016). Keyless car sharing system: A security and privacy analysis. *In IEEE International Smart Cities Conference (ISC2 2016)*, pages 1–8.

Thellmann, P. (2018). Blockchain is the infrastructure for a new decentralized sharing economy. URL: https://hackernoon.com/blockchain-is-the-infrastructure-for-a-new-decentralized-sharing-economy-f715da32bece, last checked on 2018-12-5.

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger byzantium version. URL: https://ethereum.github.io/yellowpaper/paper.pdf, last checked on 2018-08-5.