PePTCAP: A Privacy-enhancing Protocol for (Temporary) Car Access Provision

Iraklis Symeonidis, Abdelrahaman Aly, Mustafa A. Mustafa, Bart Preneel

COSIC and imec, KU Leuven
first.last@esat.kuleuven.be

Abstract. This paper proposes a novel physical keyless car sharing protocol that allows users to share their cars conveniently. In spite of many advantages, keyless car sharing systems come with substantial security and privacy challenges. Within this work, we design a concrete decentralised and Privacy-enhanced Protocol for (Temporary) Car Access Provision namely PePTCAP. PePTCAP uses multiparty computation based on Shamir's secret sharing scheme which allows the generation, update, revocation and distribution of an access token for a car in a privacy-preserving manner. In order to solve disputes and to deal with law enforcement requests, our protocol provides forensic evidence of car incidents based on threshold secret sharing. We perform a security and privacy analysis of PePTCAP followed by a complexity analysis, practical efficiency and time estimations for the multiparty computation elements of PePTCAP under realistic scenarios.

1 Introduction

Physical Keyless car Sharing Systems (KSSs) allow users to share their cars without the requisite of exchanging physical keys. Car owners' can rather use their portable devices such as smartphones to distribute (temporary) digital car keys and access tokens to other users. As such KSSs allow users to share cars more conveniently, several car manufacturers such as Volvo [1], BMW [2] and Toyota [3] have started investing in them. In contrast to traditional car rental companies, KSSs can provide a relatively inexpensive alternative to users who occasionally need a car, presenting an alternative model for car ownership [4]. Moreover, encouraging users to use KSSs can help to decrease the number of cars, effectively reducing CO_2 emissions [5] and the need for parking space.

In spite of these advantages, KSSs can introduce several security and privacy concerns [6]. For example, an adversary may try to impersonate a car owner, to tamper with the car sharing details, or even to deny having used the car. Regarding users' privacy, the adversary may try to eavesdrop and passively collect information exchanged within the car sharing system and try to infer users' behaviour [7]. For example, an adversary may try to infer the users' sharing preferences, free time activities and circle of trust. These preferences can be established by collecting information about sharing patterns such as rental time, duration, pickup, drop-off location, when, where, and with whom someone is

sharing a car. An adversary may even attempt to infer sensitive information about users such as their health status, by identifying users who use cars for disabled passengers or visit hospitals regularly.

There is a prior work on security and privacy enhanced applications of connected cars. Troncoso et al. [8] proposed a pay-as-you-drive scheme that enhances the location privacy of drivers' by sending only aggregated data to insurance companies. Balasch et al. [9] proposed an electronic toll pricing protocol where a car's on-board unit calculates locally the driver's annual toll fee while disclosing a minimum amount of location information. Mustafa et al. [10] proposed an anonymous electric vehicle charging protocol with billing support. Also, EVITA [11] and PRESERVE [12] are research projects focusing on designing and prototyping onboard units for communication systems of connected cars with privacy in mind. However, apart from the comprehensive security and privacy analysis of KSS performed by Symeonidis et al. [6], there is no prior work addressing the security and privacy issues in such systems. To the best of our knowledge, this is the first work that proposes a concrete protocol to address the identified issues.

One way to mitigate the security and privacy issues is to have a peer-to-peer protocol between both the users and the car. The car owner can generate a (temporary) access token for her car using the car key and distribute it to the other user, the consumer, who can use the token to access the car. However, this approach has two main limitations: (i) the owner and the consumer may not trust each other, thus affecting the accountability of the system, and (ii) the owner has to have a copy of the car key on her personal device which is prone to get lost or stolen. These limitations can be overcome by having a centralised entity which is trusted by both users, and which performs the access token generation on behalf of the car owner. However, such a centralised entity will have to be fully trusted, which might not be realistic under real world scenarios. Moreover, this entity can jeopardize users' privacy as it will have access to users' booking details, and the cars' keys. To mitigate all these concerns, we propose to use secure Multiparty Computation (MPC) in combination with a public ledger for generating and delivering the access token to the consumer. A detailed argumentation for the system model of PePTCAP can be found in Appendix A.

Contributions. Our contributions are twofold: (i) We design a concrete decentralised and privacy-preserving protocol for a (temporary) car access provision using MPC namely PePTCAP. Our protocol also covers an update and revocation operation upon a mutually agreed modification of the booking details or a misbehaving consumer. For dispute resolution, we base our protocol on threshold secret sharing in order to provide forensic evidence of car incidents at the request of law enforcement. (ii) We perform a security and privacy analysis of PePTCAP and evaluate it from a theoretical complexity and practical efficiency under realistic scenarios.

Outline. The rest of this paper is organised as follows: Sections 2 and 3 provide the preliminary information and cryptographic building blocks used in the design

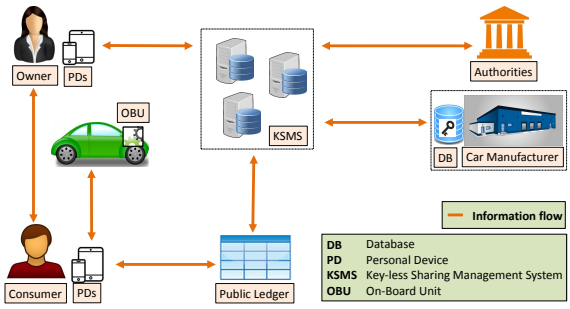

Fig. 1. System model of a physical Keyless car Sharing System (KSS).

of PePTCAP, respectively. Section 4 describes PePTCAP in detail. Section 5 provides the security and privacy analysis of PePTCAP. Section 6 evaluates its theoretical complexity and practical efficiency. Section 7 concludes our work.

2 Preliminaries

Within this section, we describe the system model, the entities involved and the functionalities of a KSS. Moreover, we specify the threat model, assumptions and the security and privacy requirements which PePTCAP needs to satisfy.

System Model. As shown in Fig. 1, KSS consists of the following entities [6]. *Users* are individuals who are willing to share their cars, *owners*, and use cars which are available for sharing, *consumers*. Users' *Portable Devices* (PDs) are mobile devices such as smartphones. An *On-Board Unit* (OBU) is an embedded or standalone hardware/software component which is part of the access management system of a *car*. It has a wireless interface such as Bluetooth, NFC or LTE. The *Car manufacturer* is responsible for generating and embedding a digital key into each car. These keys are used for car sharing and are stored in the manufacturers' *Database* (DB). The *Keyless Sharing Management System* (KSMS) is a complex of *servers* that help owners with car access token generation, distribution, update and revocation. The access token is published in a *Public Ledger* (PL), which is a publicly available source of information. In the case of a dispute, law enforcement *authorities* can assist users in resolving it.

Once users (and their cars) have been registered within the KSS, users could post their offers and requests for sharing cars. If two users agree on the sharing (*booking*) details, an owner can initiate a procedure for generating and distributing a temporary access token for the consumer to access the car. Moreover, the owner can also update or revoke the generated access token. When the consumer receives an access token, she/he can use it to authenticate to and access the car.

Threat Model and Assumptions. Owners, car manufacturers, and the KSMS are honest-but-curious entities. They will perform the protocol honestly but they

are curious to extract private information about users. Consumers and outsiders can be malicious. Cars are trusted entities whereas, users' PDs are untrusted. Within the KSS, we make the following assumptions. Each entity has a private/public key pairs and a corresponding digital certificate. Entities have copies of each others' certificates. For PePTCAP we assume that the communication channels among entities are secure and authenticated such as using SSL-TLS [13]. Moreover, an adversary cannot break the underlying cryptographic primitives.

Protocol design requirements. PePTCAP should satisfy the following security and privacy requirements.

- *SR1 - (Temporary) car access provision: No one, but the owner, should be able to provide a consumer with (temporary) access to her car.*
- *SR2 - The smallest number of trusted entities having access to a car key: No one, but the car and car company, should hold the car keys.*
- *SR3 - Booking details data authentication: The car should verify the origin and integrity of the booking details.*
- *SR4 - Backward and forward secrecy: Compromise of a key used to encrypt an access token should not compromise other tokens published on the PL.*
- *SR5 - Non-repudiation of origin and receipt of an access token: The owner should be able to prove that an access token was published on the PL and delivered to the consumer.*
- *PR1 - Booking details confidentiality: No one, but the car, should access the booking details.*
- *PR2 - Consumer's anonymity: No one, but the shared car, should learn the identity of the consumer.*
- *PR3 - Consumer's unlinkability: No one, but the shared car, should be able to link two booking requests of the same consumer.*
- *PR4 - Owner's anonymity: No one, but the KSMS, learns the identity of the owner.*
- *PR5 - Owner's unlinkability: No one, but the KSMS, should be able to link two booking requests of the same owner.*
- *PR6 - Undetectability of an access token operation: No one should be able to distinguish between an access token generation, update and revocation.*
- *PR7 - Forensic evidence provision: The KSMS should be able to provide authorities with forensic evidence of car incidents at law enforcement requests.*

3 Cryptographic building blocks

3.1 Security Model for Multiparty Computation

Shamir's secret sharing [14] can be used in combination with the multiplication protocol introduced by Ben-or et al. [15], later improved by Gennaro et al. [16]. Additionally, sharing mechanisms such as Shamir's sharing primitive, provide non-interactive addition and scalar multiplication for secretly shared

values. Furthermore, the results by Bern-or et al. in [15] (commonly referred to as BGW) and Chaum et al. [17] have proven that it is possible to calculate any function with perfect security in the presence of active and passive adversaries under the information-theoretic model. However, both protocols are bounded by the threshold properties of Shamir's scheme. Perfect security can be achieved only under an honest majority (1/2 for passive and 2/3 for active adversaries). The protocol assumes the use of private channels in between the KSMS servers and verifiable secret sharing (VSS) for the malicious case.

Although these threshold properties might sound as a limitation for some applications, i.e., dishonest majorities, this is not our case. By using protocols such as BGW, a majority coalition can reconstruct the secrets without the other parties' inputs. This kind of collusion is a desired property of our system. Note that a 3 party protocol should allow 2 computational parties to reconstruct the secret in exceptional cases. Our protocol, however, is *MPC-agnostic*, this means that it does not depend on the protocols that implement the MPC functionality (as long as addition and multiplication are provided). The protocol could well be executed using more recent MPC protocols that are secure against dishonest majorities such as SPDPZ [18], BDOZ [19] or MASCOT [20]. Typically these protocols work over a finite field limited by some q, we denote this field $\mathbb{Z}_q$. We assume the q to be a sufficiently large prime number or RSA modulus.

3.2 Additional multiparty constructions and functionality

PePTCAP uses the following cryptographic functionalities for MPC:

- $[\mathtt{x}] \leftarrow \mathtt{share}(\mathtt{x})$: is used to secret share a private input. We implemented with Shamir secret sharing primitive [14].
- $\mathtt{x} \leftarrow \mathtt{open}([\mathtt{x}])$: reconstructs the private inputs based on the secret shares.
- $[\mathtt{z}] \leftarrow \mathtt{add}([\mathtt{x}], [\mathtt{y}])$: uses the properties of Shamir's scheme to linearly add two shared inputs.
- $[\mathtt{z}] \leftarrow \mathtt{mult}([\mathtt{x}], [\mathtt{y}])$: multiplies two secretly shared inputs using the improved BGW multiplication protocol by Gennaro et al. [16].
- $[\mathtt{z}] \leftarrow \mathtt{eqz}([\mathtt{x}], [\mathtt{y}])$: secretly compute the equality test between shared inputs. In this case, this is equivalent to computing $[z] \leftarrow [x] \stackrel{?}{=} [y] \; \forall \; [z] \in \{0, 1\}$.
- $[\mathtt{r}] \leftarrow \mathtt{rand}(\mathbb{Z}_\mathtt{q})$: returns a random value bounded by q. A random number can be generated without the need of a communicational round [21].

On Secure Equality Tests: The `eqz` functionality can be implemented using a constant number of communication rounds between the computational parties. Moreover, either perfect or statistical security can be achieved by existing protocols. We refer the reader to the constructions presented in [22–24] for further details on their implementation and inner working. Although our protocol is *agnostic* towards the protocol selection for the equality tests, we assume, for simplicity reasons, that it makes use of the results by Catrina and Hoogh [24].

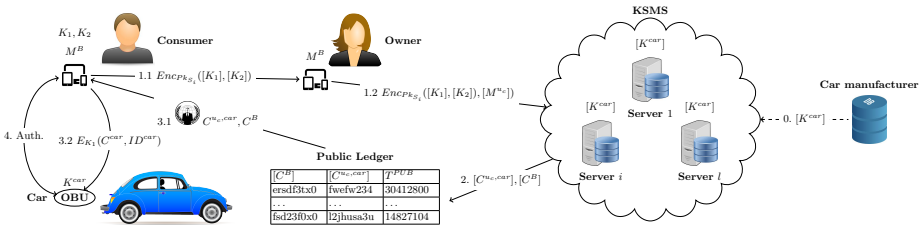

Fig. 2. PePTCAP high level overview.

AES with MPC: Research has been put forward for the implementation of AES using MPC, where the computational parties hold a secretly shared key, together with a secretly shared message. The product of the operation could be adapted to be a secretly shared AES encrypted ciphertext. We refer the reader to [25–28] for further details and treatment on the state of the art. Note that our protocol is *agnotistc* with respect to the underlying implementation of AES.

4 The PePTCAP

This section provides a detailed description of PePTCAP which consists of access token generation and distribution, as well as an update, revocation and accusation operation. Before delving into details, we give an overview of PePTCAP.

4.1 High Level Description

Below, we briefly describe PePTCAP, the steps are also shown in Fig. 2. For simplicity, the owner's and consumer's PDs are referred as owner and consumer.

Starting from the prerequisite steps, upon registration of a user and her car, the KSMS requests the car's symmetric key from the corresponding car manufacturer. The manufacturer retrieves the key from its DB, generates l secret shares of the key (assuming that the KSMS has l servers) and forwards each key share to the corresponding KSMS server. Each server stores its share of the car key in its local DB. With a car sharing request, an owner and a consumer agree on the booking details such as time period of access and car location for sharing a car. Once these details have been agreed upon, our protocol can commence.

First, the owner requests the consumer to generate two session keys: the first one is used by the KSMS to encrypt the access token and the second one for generating an authentication tag for the booking details. The consumer constructs l secret shares of each key, encrypts them with the corresponding server's public key and sends the ciphertexts to the owner. Meanwhile, the owner generates a signature of the booking details and constructs l secret shares of them and the signature. Finally, the owner sends a ciphertext of the shares of the consumer's keys and a share of the booking details and the signature to each server.

Each KSMS server queries and extracts its share of the car key from its DB. To create an access token and to guarantee the confidentiality of the sharing details, the servers encrypt the booking details using their shares of the car key. This token along with the car identity should be given only to the consumer and should be confidential against any other entity. To achieve this, a second encryption is performed; the servers encrypt the access token using their shares of the first consumer's key. Then, this encrypted access token is published on the PL. To allow the consumer to verify the booking details contained in the token, the servers generate a hash value of the booking details and encrypt this value using the shares of the consumer's second session key. Finally, each server sends to the PL its share of the encrypted access token and the encrypted hash value.

Upon receiving all the shares, the PL reconstructs both ciphertexts, publishes them and notifies the KSMS. The notification includes the time-stamp of publishing; it is also forwarded by the KSMS to the owner. Subsequently, the owner forwards it to the consumer. Upon receiving the notification, the consumer uses the received time-stamp to query the PL for both ciphertexts. To protect the consumer's anonymity, this query is made only via an anonymous communication channel. The consumer then hashes the booking details that she knows, encrypts the result with the second session key, and compares the computed ciphertext with the ciphertext retrieved from the PL, i.e., the encrypted hash value. If both are the same, the consumer is assured that the encrypted access token contains the booking details agreed during the booking phase. Finally, the consumer decrypts the other retrieved ciphertext using his first session key to obtain the access token and delivers it to the car. Upon the reception of the access token, the car decrypts it using the car key, verifies the owner's signature and accepts the token. Then, when the consumer requests to access the car, the car uses a challenge-response protocol to authenticate the consumer.

Note that owners submit private inputs to the KSMS consisting of multiple servers that function as evaluators. The number of evaluators depends on the application. There could be as many as the number of parties involved in the computation. However, this is costly in terms of performance. Thus, we assume three computational parties: one comes from the drivers' association, another from the car manufacturers' association and the third one from a control agency.

4.2 Detailed Description

PePTCAP consists of three steps: *consumer session keys generation and data distribution*, *access token generation* and *access token distribution and verification*. Before describing them in detail, we give a brief description of the *prerequisite*, and complete the section with an overview of the possible operations after PePTCAP: *consumer access to the car* and *access token update and revocation*.

Prerequisite. Before PePTCAP can commence, two prerequisite steps need to take place: *car key distribution* and *car booking*. *Car key distribution* takes place immediately after an owner, u_o, has registered her yth car, $ID_y^{car_{u_o}}$, with the

Table 1. Notation.

Symbol	Description
u_x, u_o, u_c	User x, owner, consumer
ID^B, ID^{u_o}, ID^{u_c}	ID of booking, u_o, u_c
$Cert^{u_c}$, L^{car}	Digital certificate of u_c, car's location
$\mathcal{CD}^{u_c}$, AC^{u_c}	Set of conditions, access rights under which u_c is allowed to access a car
S_i, DB^{S_i}	ith MPC Server where $i \in \{1 \dots l\}$, Database that S_i holds
$\vec{D}^{u_o}$	Car records of u_o extracted from DB^{S_i} where $\lvert\vec{D}^{u_o}\rvert = n$
$\vec{D}^{car}$, $\vec{D}^{car}_y$	Equality test output, content of $\vec{D}^{car}$ at position y
$K^x_y, Pk_i/Sk_i$	yth symmetric key of the KSS entity x, Public/private key pair of S_i
$M, M^B, H(M)$	Message, Message that contains the *booking* details, Hash operation of M
$C^K \leftarrow E_K(M)$	Ciphertext generated by the Encryption function E of M with K
$D_K(C)$	Decryption of C with K
$C^{S_i} \leftarrow Enc_{Pk_i}(M)$	Ciphertext generated by the Encryption function Enc of M with the Pk_i of S_i
$Dec_{Sk_i}(C)$	Decryption of C with the Sk_i of S_i
$\sigma_i \leftarrow Sig_{Sk_i}(M)$	Digital signature generated by the Signing function $Sig()$ of M with Sk_i of S_i

KSMS. The KSMS forwards $ID^{car_{u_o}}_y$ to the car manufacturer to request the symmetric key of the car, $K^{car_{u_o}}_y$. The manufacturer retrieves $K^{car_{u_o}}_y$ from its DB, $D\vec{B}^{cc}$, and generates l secret shares of $K^{car_{u_o}}_y$ and $ID^{car_{u_o}}_y$, denoted by $[K^{car_{u_o}}_y]$ and $[ID^{car_{u_o}}_y]$, respectively. Then, it forwards each share to the corresponding KSMS server, i.e., S_i. Upon receipt of its shares, each S_i stores ID^{u_o} together with its shares, $[ID^{car_{u_o}}_y]$ and $[K^{car_{u_o}}_y]$, in its local DB, $D\vec{B}^{S_i}$. The representations of the DB of S_i and the manufacturer are shown in Fig. 3. For simplicity, in some parts of PePTCAP we use ID^{car} and K^{car} instead of $ID^{car_{u_o}}_y$ and $K^{car_{u_o}}_y$.

$$D\vec{B}^{S_i} = \begin{bmatrix} ID^{u_1} & [ID^{car_{u_1}}_1] & [K^{car_{u_1}}_1] \\ ID^{u_o} & [ID^{car_{u_o}}_1] & [K^{car_{u_o}}_1] \\ \vdots & \vdots & \vdots \\ ID^{u_o} & [ID^{car_{u_o}}_y] & [K^{car_{u_o}}_y] \\ \vdots & \vdots & \vdots \\ ID^{u_o} & [ID^{car_{u_x}}_n] & [K^{car_{u_x}}_n] \end{bmatrix} \qquad D\vec{B}^{cc} = \begin{bmatrix} ID^{u_1} & ID^{car_{u_1}}_1 & K^{car_{u_1}}_1 \\ ID^{u_o} & ID^{car_{u_o}}_1 & K^{car_{u_o}}_1 \\ \vdots & \vdots & \vdots \\ ID^{u_o} & ID^{car_{u_o}}_y & K^{car_{u_o}}_y \\ \vdots & \vdots & \vdots \\ ID^{u_o} & ID^{car_{u_x}}_n & K^{car_{u_x}}_n \end{bmatrix}$$

Fig. 3. DB representation of the ith server (left) and the car manufacturer (right).

Car booking allows u_o and u_c to agree on the sharing details, i.e., $M^B = \{Cert^{u_c}, ID^{car}, L^{car}, \mathcal{CD}^{u_c}, AC^{u_c}, ID^B\}$, where $Cert^{u_c}$ is the digital certificate of u_c, L^{car} is the pick-up location of the car, $\mathcal{CD}^{u_c} = \{Cond^{u_c}_1, \dots, Cond^{u_c}_m\}$ is the set of conditions under which u_c is allowed to use the car (e.g., restrictions on locations, time period), AC^{u_c} is the access rights under which u_c is allowed to access the car, e.g., driver, passenger, and ID^B is the booking identifier.

Step 1: Consumer session keys generation and data distribution. The consumer generates two symmetric session keys which are then delivered to the KSMS servers. The first key will be used by the servers to encrypt the access token, so that only u_c has access to it. The second key will be used to generate an

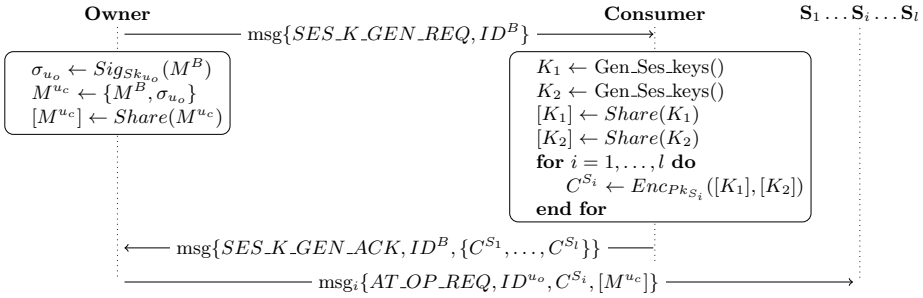

Fig. 4. Step 1: Consumer session keys generation and data distribution.

authentication tag which will allow u_c to verify that the access token contains the car sharing details agreed in the booking phase. Moreover, the car sharing details are also delivered to the KSMS. In detail, as shown in Fig. 4, u_o sends a session-keys-generation request, $GEN\_SES\_K\_REQ$, along with ID^B to u_c. Upon receiving the request, u_c generates two symmetric session keys, K_1 and K_2. Each of these keys is then transformed into l secret shares, $[K_1]$ and $[K_2]$, one for each S_i, so that none of the servers has access to these keys, but only to one share of each key. Finally, u_c encrypts the shares of the keys with the public key of the corresponding KSMS servers, e.g., $C^{S_i} = Enc_{Pk_{S_i}}([K_1], [K_2])$, so that only the servers can access the shares, before forwarding an acknowledgment message, $GEN\_SES\_K\_ACK$, together with ID^B and $\{C^{S_1}, \ldots, C^{S_l}\}$ to u_o.

While waiting for the response of u_c, u_o signs M^B with her private key, i.e., $\sigma_{u_o} = Sig_{Sk_{u_o}}(M^B)$, so that, in a later step of the protocol, the car could verify that the sharing details have been approved by the owner. She then concatenates M^B and σ_{u_o} to create M^{u_c} and transforms it into l secret shares, $[M^{u_c}]$. Upon receipt of the response of u_c, u_o forwards to each S_i an access token generation request, AT_OP_REQ, along with ID^{u_o}, C^{S_i} and the corresponding $[M^{u_c}]$.

Step 2: Access token generation. The KSMS generates the access token and publish it on the PL. Figure 5 shows the details: upon receipt of AT_OP_REQ from u_o, each S_i uses ID^{u_o} to extract $[K^{car}]$ from its database, DB^{S_i}. Initially, S_i uses ID^{u_o} to retrieve the list of all car identity and car key shares related to u_o, e.g., $[ID_y^{car_{u_o}}]$ and $[K_y^{car_{u_o}}]$. The results are stored at a vector $\vec{D}^{u_o}$ size $n \times 3$, where n is the number of cars u_o has registered with the KSMS, i.e.,

$$\vec{D}^{u_o} = \begin{bmatrix} ID^{u_o} & [ID_1^{car_{u_o}}] & [K_1^{car_{u_o}}] \\ \vdots & \vdots & \vdots \\ ID^{u_o} & [ID_y^{car_{u_o}}] & [K_y^{car_{u_o}}] \\ \vdots & \vdots & \vdots \\ ID^{u_o} & [ID_n^{car_{u_x}}] & [K_n^{car_{u_x}}] \end{bmatrix}$$

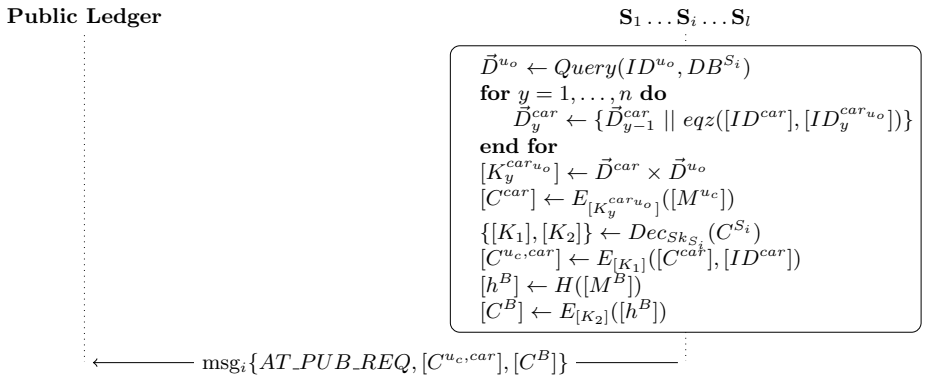

Fig. 5. Step 2: Access token generation.

To retrieve the record of the car to be shared, S_i extracts $[ID^{car}]$ from $[M^{u_c}]$ and performs a comparison with each of the n records of $\vec{D}^{u_o}$ using the `eqz()` function. The comparison outcomes 0 for mismatch and 1 for identifying the car at position y. The result of each iteration is stored at a vector $\vec{D}^{car}$, i.e.,

$$\vec{D}^{car} = [\overset{1}{0} \ldots 0 \overset{yth}{1} 0 \ldots \overset{n}{0}]$$

S_i then multiplies $\vec{D}^{car}$ and $\vec{D}^{u_o}$ to generate a third vector, i.e.,

$$\vec{D}^{car} \times \vec{D}^{u_o} = \left[ID^{u_o}[ID_y^{car_{u_o}}][K_y^{car_{u_o}}]\right]$$

from which the share of the car's secret key, $[K_y^{car_{u_o}}]$, can be retrieved. To preserve the confidentiality of $[M^{u_o}]$, each S_i encrypts it with $[K_y^{car_{u_o}}]$ to generate an access token for the car, $[C^{car}]$.

As C^{car} and ID^{car} need to be available only to u_c, a second layer of encryption is done using a session key of u_c. To retrieve the shares of the session keys, $\{[K_1], [K_2]\}$, each S_i decrypts C^{S_i} using its private key, Sk_{S_i}. Then, each S_i encrypts $\{[C^{car}], [ID^{car}]\}$ with $[K_1]$ to generate $[C^{u_c,car}]$. Next, each S_i generates an index value which will be used by u_c to identify $[C^{u_c,car}]$. It hashes $[M^B]$ to generate $[h^B]$, and encrypts $[h^B]$ with $[K_2]$ to generate $[C^B]$. Finally, each S_i sends to the PL a publishing request, AT_PUB_REQ, $[C^{car,u_c}]$ and $[C^B]$.

Step 3: Access token distribution and verification. The PL reconstructs and publishes the encrypted access token which is then retrieved by u_c. Once retrieved, the token is delivered to and verified by the car. The details are shown in Fig. 6, upon receipt of AT_PUB_REQ, $[C^{car,u_c}]$ and $[C^B]$ from each S_i, the PL uses these shares to reconstruct both, C^{car,u_c} and C^B, using the `open()` function. Then, it publishes C^{car,u_c}, C^B and T^{PUB}, which is the time of publishing, and sends an acknowledgement of the encrypted access token publication, AT_PUB_ACK, along with T^{PUB} to the KSMS (to at least one S_i). The KSMS then forwards AT_PUB_ACK to u_o who, in turn, forwards it to u_c.

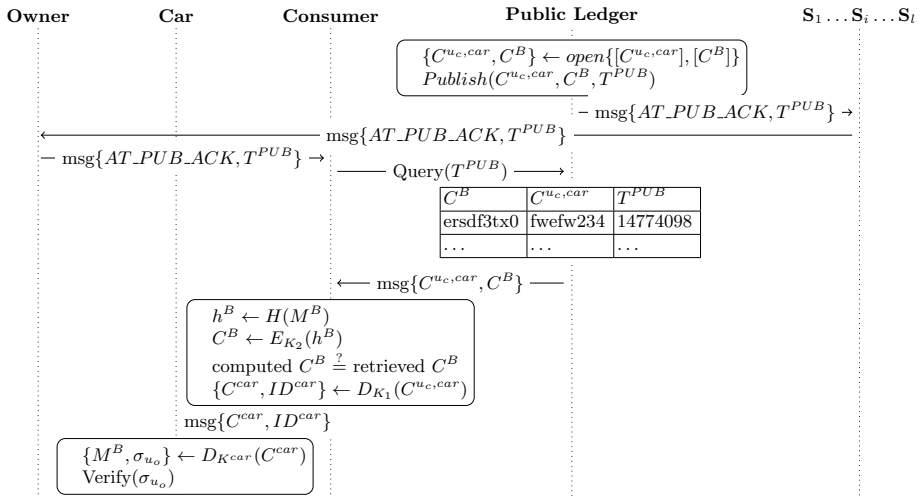

Fig. 6. Step 3: Access token distribution and verification.

Upon receipt of AT_PUB_ACK, u_c uses T^{PUB} to retrieve $\{C^{u_c,car}, C^B\}$ from the PL via an anonymous communication channel such as Tor [29], so that the PL can not identify u_c. Then, he hashes M^B that was obtained after the booking phase and encrypts the result with K_2 to compute C^B. Next, u_c checks if the computed C^B is the same as C^B retrieved from the PL. If both ciphertexts are the same, u_c is assured that the token contains the same details as the ones agreed during *car booking*. He then obtains the access token and the car identity, $\{C^{car}, ID^{car}\}$, by decrypting C^{car,u_c} with K_1, and sends them to the car.

Upon receipt of $\{C^{car}, ID^{car}\}$, the car's OBU obtains $M^{u_c} = \{M^B, \sigma_{u_o}\}$ by decrypting C^{car} with K^{car}. Then, it verifies σ_{u_o} to be assured that the booking details, M_B, have not been modified and have been indeed approved by the car owner. If the verification holds, the OBU accepts and stores M^B for later use to give u_c access to the car. The whole protocol is depicted in Appendix B.

Consumer access to the car. When u_c attempts to access the car, the car's OBU initially checks if the access attempt satisfies the conditions specified in M^B, and then verifies the identity of u_c. As the OBU has $Cert^{u_c}$ (it is included in M^B), it uses any challenge-response protocol based on public/private key [30].

Access token update and revocation. Upon an agreement among u_o and u_c to update or revoke an access token, PePTCAP can be performed as described in steps 1-3. The values of an update request can be changed according to new booking details, $\hat{M}^B$, whereas for revocation, each of the parameters in $\hat{M}^B$ can receive a predefined value indicating the revocation action. However, there are occasions when u_o may need to enforce an update or revocation of an access token. To prevent u_c from blocking such operations, PePTCAP should be exe-

cuted only by u_o, without the involvement of u_c. This is: u_o generates session keys, requests an access token, queries the PL and sends the token to the car.

5 Security and Privacy Analysis

This section analyses the security and privacy properties of PePTCAP. Note that the owner and consumer already know each others identity and the booking details. Thus, we focus only on other parties who want to learn this information.

(Temporary) car access provision. PePTCAP allows a car owner to initiate a (temporary) access token generation and distribution to her car via a consumer. This token contains various conditions under which the chosen consumer can access the car, the consumer-specific information, i.e., his digital certificate, as well as the owner's signature on these details. Once the car receives and verifies this signature, it can allow the consumer to have access to it if all the conditions in the token are met and it successfully verifies the consumer using a standard challenge-response authentication protocol. Using a token which contains car access conditions settable by the owner, coupled with the owner's signature verification at the car, allows only the owner to provide a consumer with access to her car. Thus, PePTCAP satisfies *SR1*.

The smallest number of trusted entities having access to car key. With PePTCAP, only the car manufacturer and the car itself hold copies of the car key. Although the access token for the car is generated by the KSMS servers by encrypting the booking details with the car key, none of these servers actually have access to the key, but only to a share of this key. Assuming that these servers do not collude, coupled with the fact that, by design, the car manufacturer and the car have copies of the car key, PePTCAP does not allow any additional entities to have access to car keys, thus satisfying *SR2*.

Booking details data authentication. An owner who initiates the access token generation and distribution, first signs the booking details using its private key before sending them to the KSMS in shares. Therefore, once the car receives the token and obtains the booking details, it can verify the owner's signature on the booking details. In other words, the car can verify the source of the booking details, the owner, and their integrity. Thus, PePTCAP satisfies *SR3*.

Backward and forward secrecy. The access token is encrypted with the consumer's session key before the result is published on the PL. Also, for each token, a consumer uses a freshly generated symmetric key. As long as he generates unique session keys for each request, even a compromise of these keys will not affect the rest of the data on the PL. Thus, PePTCAP satisfies *SR4*.

Non-repudiation of origin and receipt of the access token. An access token is generated by the KSMS servers and published on the PL, if the car owner has initiated the procedure by sending the necessary information to the KSMS. Moreover, once the token is published, the PL generates an acknowledgment of publication which is sent to the KSMS. The KSMS then forwards it to the owner who, in her turn, forwards it to the consumer. Assuming that (i) the communication channels between the entities are authenticated, in other words,

the KSMS verifies the owner's identity before generating the token, (ii) the consumer gets the notification from the owner who initiated the token generation, and (iii) the acknowledgement is signed by the PL, the owner can prove that he initiated the token generation and that the token was published on the PL. If the consumer is bound by his contract with the KSMS to retrieve the token from the PL once he receives the acknowledgment of publication, the owner can also prove that the consumer received the token. Thus, PePTCAP satisfies *SR5*.

Confidentiality of booking details. The owner sends the booking details in shares, which are indistinguishable, i.e., randomized, to each of the KSMS servers, so none of them has access to the booking details. Then, the servers encrypt these details twice, first with the car key and then with the consumer's keys, before the result is published on the PL. Thus, even the car manufacturer, who knows the car key, can not decrypt the ciphertext. The published ciphertext can only be decrypted by the consumer. However, the result will be the access token, i.e., the booking details encrypted with the car key. Once the consumer obtains the token, he can deliver it to the car where the car can decrypt it and read the booking details. As the owner and consumer already know these details, with PePTCAP, only the car learns them. Thus, PePTCAP satisfies *PR1*.

Consumer's anonymity and unlinkability. With PePTCAP, the only consumer-identifiable data is the consumer's certificate included in the booking details. However, as mentioned above, with PePTCAP, only the car obtains the booking details, thus only the car learns the identity of the consumer. Moreover, apart from the PL, the consumer only communicates with the owner and the car. The owner already knows the identity of the consumer from the booking step, and the car is supposed to learn his identity, so that it can perform a proper access control. Hence, as long as the consumer retrieves data from the PL through an anonymous communication channel, PePTCAP allows only the car to learn the consumer's identity. Moreover, as every booking has a new randomly generated booking identifier, only the car could link two booking sessions of the same consumer. Thus, PePTCAP satisfies *PR2* and *PR3*.

Owner's anonymity and unlinkability. Only the car reads the owner's signature on the booking details, thus no other entity could use the signature to learn the owner's identity. Moreover, the KSMS servers store the shares of the car keys in their local databases, so that they do not query on demand the database of the car company, thus avoiding the car company linking an access token generation to a specific owner. In addition, the owner communicates only with the consumer and the KSMS. As the consumer already knows the owner's identity, only the KSMS learns it from the communication. As every booking identifier is different for each booking request, also only the KSMS can link two requests from the same owner. Thus, PePTCAP satisfies *PR4* and *PR5*.

Undetectability of an access token operation. Access token generation, update or revocation is performed using the same steps and type of messages sent to the KSMS and PL. Hence, outsiders and system entities can not distinguish which operation has been requested. Thus, PePTCAP satisfies *PR6*.

Forensic evidence provision. In case of disputes, the information related to a specific transaction may need to be reconstructed. This reconstruction can be done only if the KSMS servers collude and reveal their shares. In our setting, these servers have competing interests, thus they would not collude unless provided with low enforcement requests by authorities. With PePTCAP, due to the properties of threshold secret sharing, the private inputs can be reconstructed by a majority coalition. This is, given our three party configuration, it suffices two of such parties inputs to reconstruct the secrets (for semi-honest and malicious cases). If, for example, an incident were to happen involving the car or the owner, a control agency only needs to team up with the affected party to reveal information related to the car access. Thus, PePTCAP satifies *PR7*.

6 PePTCAP Evaluation

In this section we evaluate PePTCAP from a theoretical complexity and practical efficiency point of view. We focus on the computational cost involved in the MPC operations performed by the KSMS servers. Computational cost, in our case is measured by the number of non-concurrent operations performed by the KSMS servers. We can categorize these operations in two distinctive groups: the operations that need message exchanges among the servers, and ones who can be performed locally. We call the former round-based whereas the latter local. Normally round-base operations require far more resources than local operations, so much so, literature typically address the complexity in terms of communication complexity disregarding any polynomial amount of local operations. In other words, local operations are consider to be *free*, in the sense that its computation only requires a small set of basic arithmetic operations, e.g., addition of secret shares, scalar multiplication. An example of a round-based operation is multiplication, which requires at least one communication round [15, 18]. More precisely, in our case, the improved BGW multiplication protocol by Gennaro [16] requires one round for resharing an internal state product. Our analysis takes this into account and defines complexity as the number of communication rounds performed by the KSMS servers, i.e., communication complexity. This is a typical approach taken for complexity analysis by the MPC community [22–24, 31, 32].

Theoretical Complexity: The KSMS servers first recover the car key by executing a linear exploration of an anonymity set stored in their local databases in shared form. This operation requires the execution of at least $|ID_b^{car}|$ multiplications per entry on the anonymity set D^{u_o}, where ID_b^{car} is the bitsize representation of ID^{car} and $|ID_b^{car}|$ is its size. This could be improved to the size of the anonymity set, by using the disclose if equal operation from [23]. However, this can only be achieved at the cost of privately permuting the entries in advance. Note that privately permuting a vector has a cost of $n \cdot log(n)$ round operations, where n is the size of the vector [33]. Additional $|D^{u_o}|$ secure multiplications have to be performed too. This is followed by performing two rounds of encryption on the booking details and one on their hash, using AES.

Hence, the number of round-based operations depends solely on the number of S-Boxes that are executed for the encryption. We define ν to be the number of round-based operations times the number of S-boxes (16) needed to encrypt a single 128 bit block. The number of blocks is easily calculated by dividing the bitsize of the booking details to 128. The total complexity can be expressed as

$$|D^{u_o}| \cdot |ID^{car}| + |D^{u_o}| + 2 \cdot \frac{|M^{u_c}|}{128} \cdot \nu + \frac{H(M^B)}{128} \cdot \nu \tag{1}$$

round based operations. Notice that the AES encryption of the hash of the booking details can be performed in parallel. Note that, the car owner's and consumer's task of share generation can be performed locally and secret share mechanisms like Shamir secret sharing are not intrinsically computationally heavy.

Practical Efficiency: Although PePTCAP is an AES implementation *agnostic*, we make use of the results from [25] which is a BGW-based construction. We stress that, although the state-of-the-art on this kind of AES implementations has evolved, we consider [25] because it offers computational experimentation based on BGW which is relevant for our case. Moreover, we consider a relatively large token-size, to facilitate any further analysis for smaller (more realistic) scenarios. We assume the following configuration for the booking details: $Cert^{u_c}$ is 7920 bits, ID^{car} is 32 bits, L^{car} is 64 bits, $\mathcal{CD}^{u_c}$ is 96 bits, AC^{u_c} is 4 bits, ID^B is 32 bits, and σ is 512 bits, consisting a bitsize of 8660 bits for M^{u_c}. We consider an anonymity set D^{u_o} of median size, i.e., 40, being sufficient to hide the average number of cars a single individual holds, and a typical 256 hash bitsize. Finally, following [25], we set ν to 71. Using (1), the total number of round-based operations equals 1118. However, this ν value corresponds to a highly parallelized case described in [25]. The value of ν raises to 2202 when no such parallelization on round-based operations is considered. If we use this greater ν value, the total raises to $305,188$ non-concurrent round-based operations.

We measured the amount of time needed to execute a single round-based operation, which is multiplication in our case, under BGW. We used the BGW implementation introduced in [34], and implemented in *C++*. We run 10^6 multiplication operations on a 64 bit server running 2*2*10-cores Intel Xeon E5-2687 at 3.1GHz 256 GB of RAM memory available to a Linux distribution and three parties running on the same machine. The average execution time for such operation resulted in 2.08×10^{-5} seconds. The extrapolation of this value in our case leads to an execution time equivalent to 0.0232544 seconds for the highly parallelized case, and 6.35 seconds when no such parallelization is considered. Although these running times would satisfy the needs of a realistic application, this is only a relative approximation of what the execution time for the MPC components would be. As this is only an estimation, results might vary depending instantiation details of any implementation.

7 Conclusions

Although physical keyless car sharing systems allow users to share their cars conveniently, they come with security and privacy challenges. To address these challenges, this paper proposes a decentralised and privacy-enhancing protocol for (temporary) car access provision, PePTCAP. PePTCAP uses multiparty computation based on Shamir's secret sharing scheme which allows a generation and distribution of an access token for a car in a privacy-preserving manner, as well as, dealing with disputes upon law enforcement requests. Moreover, we performed a security and privacy analysis of PePTCAP. We also performed a complexity analysis, practical efficiency and time estimations for the multiparty computation elements of PePTCAP. Our estimations show that PePTCAP is fast enough for realistic scenarios. As future work, we plan to extend PePTCAP to include additional operations such as booking and payment.

References

1. Volvo: Volvo Keyless Cars Accessed November, 2016.
2. BMW: Drivenow Accessed November, 2016.
3. TODAY, U.: Toyota will test keyless car sharing Accessed November, 2016.
4. Shaheen, S.A., Mallery, M.A., Kingsley, K.J.: Personal vehicle sharing services in North America. Research in Transportation Business & Management **3** (2012) 71–81
5. Shaheen, S.A., Cohen, A.P.: Car sharing and personal vehicle services: worldwide market developments and emerging trends. Int. Journal of Sustainable Transportation **7**(1) (2013) 5–34
6. Symeonidis, I., Mustafa, M.A., Preneel, B.: Key-less Car Sharing System: A Security and Privacy Analysis. In: 2016 IEEE International Smart Cities Conference (ISC2). (Sept 2016) 1–7
7. Uber: New App Features and Data Show How Uber Can Improve Safety on the Road Accessed July, 2016.
8. Troncoso, C., Danezis, G., Kosta, E., Balasch, J., Preneel, B.: PriPAYD: Privacy-Friendly Pay-As-You-Drive Insurance. IEEE Trans. Dependable Sec. Comput. **8**(5) (2011) 742–755
9. Balasch, J., Rial, A., Troncoso, C., Preneel, B., Verbauwhede, I., Geuens, C.: PrETP: Privacy-Preserving Electronic Toll Pricing. In: 19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings. (2010) 63–78
10. Mustafa, M.A., Zhang, N., Kalogridis, G., Fan, Z.: Roaming electric vehicle charging and billing: An anonymous multi-user protocol. In: IEEE SmartGridComm. (Nov 2014) 939–945
11. EVITA: E-safety Vehicle Intrusion Protected Applications (EVITA) Accessed November, 2016.
12. PRESERVE: Preparing Secure Vehicle-to-X Communication Systems (PRESERVE) Accessed November, 2016.
13. Group, N.W.: The Transport Layer Security (TLS) Protocol, Version 1.2 Accessed November, 2016.
14. Shamir, A.: How to share a secret. Commun. ACM **22**(11) (1979) 612–613

15. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: STOC, ACM (1988) 1–10
16. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In: PODC '98, ACM (1998) 101–111
17. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols. In: STOC, ACM (1988) 11–19
18. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: CRYPTO '12. Volume 7417 of LNCS., Springer (2012) 643–662
19. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: EUROCRYPT '11. Volume 6632 of LNCS., Springer (2011) 169–188
20. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. Technical report, Cryptology ePrint Archive, 201 6. http://eprint. iacr. org/2016/505
21. Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: TCC 2005. Volume 3378 of LNCS. Springer (2005) 342–362
22. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: TCC 2006. Volume 3876 of LNCS., Springer (2006) 285–304
23. Lipmaa, H., Toft, T.: Secure equality and greater-than tests with sublinear online complexity. In: ICALP (2). (2013) 645–656
24. Catrina, O., de Hoogh, S.: Improved primitives for secure multiparty integer computation. In: SCN. (2010) 182–199
25. Damgård, I., Keller, M.: Secure multiparty AES. In: International Conference on Financial Cryptography and Data Security, Springer (2010) 367–374
26. Damgård, I., Keller, M., Larraia, E., Miles, C., Smart, N.: Implementing AES via an Actively/Covertly Secure Dishonest-Majority MPC Protocol. In: Security and Cryptography for Networks. Volume 7485 of LNCS., Springer (2012) 241–263
27. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer (2015) 430–454
28. Grassi, L., Rechberger, C., Rotaru, D., Scholl, P., Smart, N.P.: MPC-Friendly Symmetric Key Primitives. Technical report, Cryptology ePrint Archive, Report 2016, 2016. http://eprint. iacr. org
29. Feamster, N., Dingledine, R.: Location diversity in anonymity networks. In: Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, WPES 2004, Washington, DC, USA, October 28, 2004. (2004) 66–76
30. Diffie, W., van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. Des. Codes Cryptography **2**(2) (1992) 107–125
31. Zhang, Y., Steele, A., Blanton, M.: Picco: a general-purpose compiler for private distributed computation. In: Proceedings of the 2013 ACM SIGSAC conference on Computer; communications security. CCS '13, New York, NY, USA, ACM (2013) 813–826
32. Aly, A., Cuvelier, E., Mawet, S., Pereira, O., Van Vyve, M.: Securely solving simple combinatorial graph problems. In: Financial Cryptography. (2013) 239–257
33. Czumaj, A., Kanarek, P., Kutylowski, M., Lorys, K.: Delayed path coupling and generating random permutations via distributed stochastic processes. In: SODA '99, SIAM (1999) 271–280

34. Aly, A.: Network Flow Problems with Secure Multiparty Computation. PhD thesis, Universté catholique de Louvain, IMMAQ (2015)
35. Council of the EU Final Compromised Resolution: General Data Protection Regulation. http://www.europarl.europa.eu Accessed Feb, 2015.

Appendix A

Straw-man arguments

We outline two physical Keyless car Sharing Systems (KSSs), one is based on a peer-to-peer and the other on a centralised architecture. Moreover, we argue why these architectures are not good solutions. In short, the peer-to-peer KSS architecture fails to provide the *non-repudiation* property, and it is weak against adversaries who can physically steal the owner's portable devices and extract the car's symmetric key stored on them. The centralised KSS architecture fails to provide user's *anonymity* and *unlinkability* as the Keyless Sharing Management System (KSMS) can link all the booking sessions and construct a detailed user's profile (profiling) [35]. Next we collaborate more on these arguments.

A peer-to-peer KSS architecture. We first consider a simple peer-to-peer protocol that consists of an owner, a consumer and a car. The car owner holds the key of the car and have it stored in his portable device. Moreover, an owner and a consumer have agreed upon the booking details for sharing a car. For a car sharing request, the car owner can generate a (temporary) access token for her car using the agreed booking details and the key of the car. Upon generation of the access token, the owner distributes it to the consumer, who can use the token to access the car. This architecture has two weaknesses: *weak non-repudiation* and *weak car key protection*.

Weak non-repudiation. In a peer-to-peer KSS architecture, providing an accountability might be challenging as resolving disputes will have to rely completely on the traces and evidences left on the car or on both users' portable devices. However, considering that users might be distrusting each other, coupled with the fact that both users will have physical access to the car, they might attempt to manipulate or even completely destroy the evidences on the car or their devices. Moreover, the car is a mobile entity which might not all the time have a connectivity with its owner. It can also be the case where an adversary (consumer) have the means to deny the car reporting to the owner the use of the car. The consumer might aim to disrupt any communication between the car and the owner's device or to physically violate and brake the car's OBU. To better mitigate such physical attacks and provide a strong non-repudiation property, a trusted entity operating in a physically secure environment is essential for a fair dispute resolution within the KSS.

Weak car key protection. Another major weakness of the peer-to-peer KSS architecture is the fact that the symmetric key of the car has to be stored in the owner's devices. Unfortunately, such portable devices can easily get broken or even stolen. For a capable adversary, it might be feasible to extract the symmetric car key from the device by bypassing the device's security authentication mechanisms such as PIN or password. Once an adversary gains access to the car key, he/she can generate valid car sharing access tokens, so that he/she can steal the car or share it with others. To mitigate such a limitation, the smallest number of trusted entities should have access to the car key. Considering that the car manufacturer is the trusted entity which generates and embeds a key into each car, the manufacturer and the car already have copies of the car key. Thus, ideally, a KSS solution should not allow any other entities to have copies of the key.

A centralized KSS architecture. To overcome the aforementioned limitations, a centralised architecture including a Keyless Management System (KSMS) is necessary. The KSMS will deal with all the requests from the users, keep evidences for each of the users' actions (for quick disputes resolutions), as well as, hold a copy of the car, so that (i) it can generate access tokens and (ii) owners' device do not have to hold copies of car keys. However, the KSMS will have to be filly trusted by both, owners and users. Hence, such an KSS architecture also has two weaknesses: *single point of failure* and *user privacy violation.*

Single point of failure. Having a single entity managing all the access token generation requests concentrates the system computations on one entity, the KSMS, thus making it a performance bottleneck of the KSS, and an easy target for denial-of-service attacks. Hence, distributing the computational burden among several parties, for example several servers managed by different parties, could increase considerably the reliability of the KSS.

User privacy violation. Another weakness of the centralised KSS entity is that the centralised entity can jeopardize the users' privacy. The KSMS might passively collect the car sharing details such as who is sharing, which car, with whom, at which location, when and for how long. It may also attempt to learn the location and availability of a car, whether a user is absent from home and whom he/she is traveling with. Moreover, the user's car sharing preferences might be inferred by a systematic collection of the user's information by the KSMS. Thereby, an adversary may infer the car owners' sharing preferences, free time activities and circle of trust. These preferences can be established by collecting information about sharing patterns such as rental time, duration, type of car, pickup, and drop-off location, how often a car is shared, and with whom. An adversary may even attempt to infer sensitive information about users such as their health status, by identifying users who use cars for disabled passengers, or regular hospital visits. Profiling constitutes a high risk for users' privacy [35]. Therefore, users' *anonymity*, booking *unlinkability* and access token *confidentiality* should be provided by any KSS.

To mitigate all these accountability, trust and privacy issues, we use a KSS architecture based on secure Multiparty Computation (MPC) in combination with a Public Ledger for guaranteeing user *accountability*, *anonymity*, *unlinkability* and access token *confidentiality*. Moreover, our system outsources the computationally heavy operation to a complex of MPC servers, bearing users from performing such operations.

Appendix B

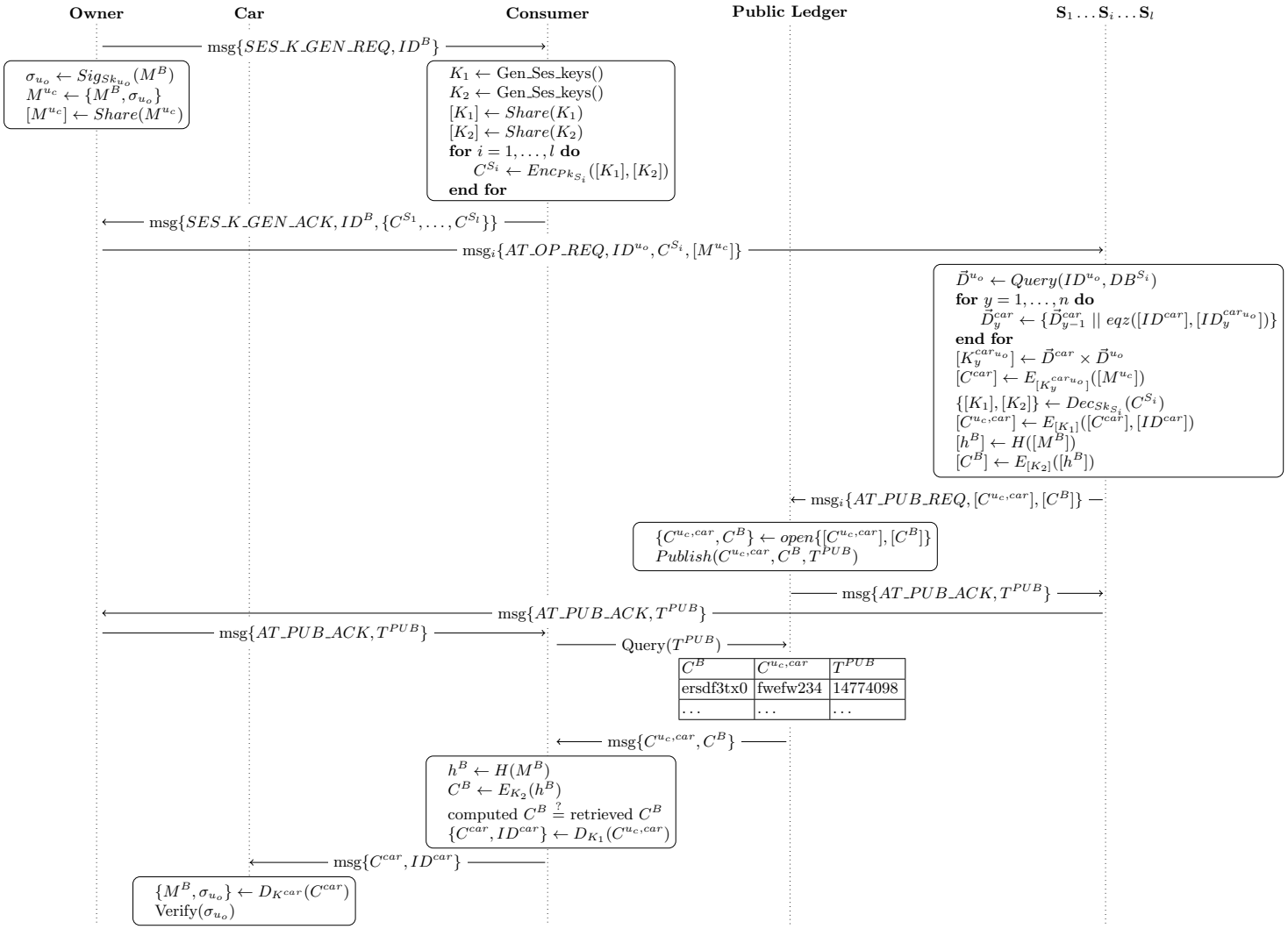

Fig. 7. PePTCAP complete representation.