Solving the Security Problems of Free-Floating Car Sharing

Nicholas R. Bushway
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
bushw011@morris.umn.edu

ABSTRACT

Free-Floating car sharing is an emerging approach to ride-sharing where people can rent cars (or share their own) for a short period of time using tracking technology to locate available vehicles nearest to them. This kind of system has a number of security risks associated with it, such as making sure users cannot be impersonated, or that malicious users cannot gain access to somebody's car.

This paper will be analyzing the security-related issues that must be solved to have an effective free-floating car sharing system. This will be done by establishing a high-level overview of a proposed car-sharing system, as well as the necessary security requirements of such a system. The paper will then go over two different proposed systems that utilize existing security protocols to address the established security requirements of a free-floating car sharing system.

Keywords

Car Sharing; Cryptography; Security

1. INTRODUCTION

Individuals living in large cities are finding it less worth it to own a vehicle. The time spent driving it is minimal, and one often has to pay a monthly fee for parking. For this reason, people are beginning to seek out alternative solutions for the times they need to use transportation. With a growth trajectory of 34% for the car-sharing market from 2016-2024[1], this desire for alternative transportation solutions can be seen.

Free-floating car sharing is a form of car sharing that allows users to make use of tracking software to "book" vehicles that are near their location (typically with an application on the user's mobile phone)[6]. Though free-floating car sharing has not been widely implemented in the United States, there are a number of free-floating car sharing services appearing, such as BMW's DriveNow in Germany, or Evo Car Share in Canada. The more available these systems become, it is possible that the number of individuals who choose to own cars in the future may decrease significantly. With this in mind, it is necessary that these systems are safe and secure for the people using them.

This paper lays out how we ensure that free-floating car sharing systems accomplish these secure properties through proposed models and subsequent systems.

UMM CSci Senior Seminar Conference, April 2019 Morris, MN.

2. BACKGROUND

Free-floating car sharing systems requires a number of different building blocks to be effective in protecting users' identity as well as their vehicles.

2.1 Security Concepts

The first of these building blocks are *authentication protocols*. Authentication is used by a server in order to obtain information regarding who a user is, as well as proof that this user is who they are saying they are. This is coupled with *authorization*, which is the act of ensuring a user is allowed to access something.

When we want to keep a piece of data hidden from others we employ *encryption*. When we encrypt data, we transform it from its readable form (*plaintext*), to an indecipherable form (*ciphertext*) with some type of cryptographic key. When we turn the data back into plaintext, we *decrypt* it. The most basic type of key would be a *symmetric key*, which is a key that essentially works in the typical way we think of keys. A symmetric key used to encrypt data is the same key used to decrypt it. If we encrypt a piece of data with this type of key, we can give someone else a copy of this same key if we want them to be able to access it.

An example of encryption in use in an online system would be a Secure Sockets Layer (SSL) protocol. The SSL protocol is the standard method for keeping information transferred between two entities (this could be two servers, a client and a server, two clients via a central server, etc.) protected and unreadable to any other entity. This is most typically done by installing a server-side *certificate*, which is a small file installed onto a web server that serves to both verify the identity of a web page and encrypt any data transferred to or from the web-server.

An SSL protocol utilizes *public key cryptography*, which is a method of encryption that uses two keys: a *public key* and a *private key*. These two keys are a mathematically related pair. What is encrypted by one can only be decrypted by the other. Which key does what depends on the task being performed. Public keys can be freely shared, whereas private keys are known only to the owner. If we are receiving a piece of data we want hidden, we can give the sender a copy of our public key to encrypt that data, so that we are the only ones who can access it via our private key.

Another method of public key cryptography would be *digital signatures*, which are the encrypted cipher text of a piece

of data, sent along with the plain text in order to determine the origin of that data. The way it works is that the sender would use their private key to encrypt the signature (also known as *signing* the data). This way, if the recipient is able to decrypt the signature with the sender's public key, it is ensured that the data came from that sender.

A necessary component in cryptography are *cryptographic hash functions*. Hashing is a mathematical operation performed on a piece of data that irreversibly converts it into a unique piece of text. You could input something to a hash function and see the result, but you would not be able to decipher the original input from any unique hash [3]. Hashed data is mainly different from encrypted data due to the lack of any key that could be used to decipher it. This is used when you create any kind of online profile attached to a password. The service would ideally not store the password itself, but instead the hash of that password. When the user goes to input their password, the hash of that input can be checked against the stored hash to determine if the entered password is correct.

A key use of cryptographic hash functions is *message authentication*. Message authentication is fairly similar in functionality to digital signatures, though rather than one party having a private key and the other a public key, both individuals have a copy of the sender's key. The sender creates a hash of a pairing of the message and the key (denoted the *authentication tag*). The sender sends the plaintext message along with this authentication tag to the other party. The recipient can then use the plaintext message along with the previously possessed key to verify the integrity of the tag.

Secret Sharing is a method employed on data intended to be kept secret where it is shared among a group of entities, who each receive a *share* of the secret. These shares are distributed in such a way that if there are n total shares, any entity with fewer than all n shares has no more information than someone with 0 shares [7]. Once the n shares are put back together the original secret can be obtained.

2.2 Analysis Concepts

A *threat* is anything that can potentially cause harm to any system related to computers. Similarly a *vulnerability* is an area of flaw in a system that would allow for a threat to turn into actual damage being done against the system. This could be sensitive information being exposed, or a malicious user bypassing necessary authentication/ authorization steps to get to something they should not have access to. Threats and vulnerabilities together make up the aspects of a system that must be acknowledged and properly assessed in order to avoid disaster.

A *threat model* is a way that we can establish the threats facing a system. This includes the components that could possibly contain vulnerabilities and how they may be exploited from the perspective of an attacker.

3. HIGH LEVEL OVERVIEW

To establish the security problems facing free-floating car sharing, a high-level overview of a car sharing system along with the security requirements must be established. Symeonidis et al. propose a high level model called the *keyless sharing system* as well as something called a *keyless sharing management server*. As the name suggests, it proposes a system for car sharing where physical keys are not used. The reason for this is that the exchange of keys would add time and inconvenience to any given transaction (and would make actual theft dangerously easier). That being said, eliminating the use of physical keys is an additional constraint that introduces more potential security related concerns.

3.1 Components of a Keyless Car Sharing System

The keyless car sharing system proposed in the paper has a number of necessary components to it. These components introduce concepts that will be shared in all solutions within this paper. This includes:

- *Users*, that consist of *consumers* seeking to rent a car for use, and *owners*, who own a car and are willing to rent it out to a consumer for a period of time.

- Keyless Sharing Management Server (KSMS). This is either the central server (or a number of interacting servers) where car booking is handled (finding nearby cars and the interaction between consumer and owner), "access rights" are distributed and revoked when necessary. It is also where general user authorization takes place regarding what car a user has access to, and if they still have access to it.

- *Keyless Sharing On Board Unit* (KS-OBU) is the physical device that will actually be installed onto all cars used within a car sharing system. It will have wireless functionality (either Bluetooth, WiFi, or LTE)[5] that would work in regards to the system's access management for the vehicle itself.

- *Keyless Sharing Application* (KSApp) would be some type of web/mobile app that the users use to interact with eachother and handle car booking from an interface.

- *Portable device* is a device that people use to access the KSApp and overall car sharing system.

Figure 1 can be seen as a visual aid for the interaction between these components.

3.2 Threat Model

In this section we establish a threat analysis of a keyless car sharing system via a threat model. Users are defined as untrustworthy/ malicious. This does not mean that users have inherently ill-intent or anything. It simply means that it is possible for users to have these motives, and if they do it must be considered what they could be capable of. What users could be capable of is collecting and then manipulating information stored within the system in order to jeopardize the information that owners and consumers receive.

The KS-OBU is defined as "untrustworthy but tamper-evident" [5]. What this means is that the KS-OBU contains data that could compromise the user's privacy such as the vehicle's location, or where it's going. However, the tamper-evident part means that the device has security measures in place in order to prevent the user's privacy from being compromised. This involves storing cryptographic keys, performing cryptographic operations, as well as the functionality to regularly roll out software updates in order to keep up with newly discovered breaches to keep the users (both consumer and owner) safe.

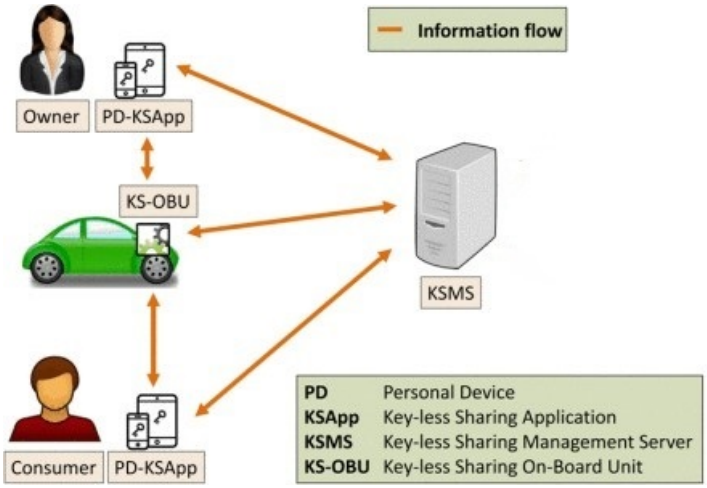

Figure 1: A visual overview of the KSS [5]

The KSApp is also defined as "untrustworthy but tamper evident"[5]. This component would be equipped with security measures in order to protect against data breaches and malware. This would be in the form of encrypting and storing user's sensitive information such as passwords or private keys. Furthermore, access to this mobile app would require authentication steps in order to ensure that the individual accessing it is actually the established user.

The KSMS is defined as "honest-but-curious or even semi-honest."[5] This means that both of these entities may try to access information from the KSS in order to gather booking details (or preferences), who they are booking with, or how often they are sharing their car. However, the paper assumes that these entities will not attempt to tamper with the data received from the KSS.

Finally, we also need to protect the system from any third party that may attempt to eavesdrop, or modify, data transferred among any participants, or try to disrupt service.

3.3 Threat Analysis

There are a number of different threats that can face the KSMS.

- *Spoofing*: This is when a malicious user takes on the identity of a different individual that another user trusts, and attempts to use this fake identity for malicious gain.

- *Tampering with Data*: This is the act of an adversary changing data stored within the KSS, such as profile information of a user, availability information of a car, details of a car (such as year, number of seats) in order to manipulate others, or to just undermine the integrity of the KSS.

- *Information Disclosure*: If information stored within the KSS is not kept confidential, an adversary could disclose the information of another user. This could include sensitive profile information (access keys, personal information), or booking details such as vehicle location to a party that should not have this information.

- *Repudiation*: Information exchanged between parties must be non-repudiable. The definition of repudiate is to deny. Non-repudiable information means that a party should not be able to deny that they performed a certain action or agreed to certain booking details. This could result in monetary disputes regarding distance traveled, someone agreeing to (and paying for) a booking that they now deny doing, etc. Non-repudiation is necessary to be able to resolve these disputes effectively.

- *Denial-of-Service* (DoS): A DoS attack is an attempt to make a service inaccessible to others. This could be done by targeting one of the components within the KSS, such as the KSApp, or the KS-OBU in order to make that component unable to perform network operations. For this reason the various components should have anti-malware software to safeguard against these attacks.

- *Elevation of Privilege*: This would be when a user elevates their access to resources within the KSS, or similarly bypasses authorization steps in the software to access data they are not allowed to access.

3.4 Security Requirements

With the noted security threats, there are a number of requirements that the software of a KSS must meet in order to curb the possibility of these threats arising, and minimize vulnerabilities.

The first security requirement is *Entity Authentication*. This "assures to an entity that the identity of a second entity is the one that is claiming to be"[5]. This would be done by having steps that require entering a password and/or pin before sensitive steps during any process within the KSS. This also entails the requirement of access tokens to use the different components of the KSS, such as the KSApp, and the KS-OBU. This would allow prevention of spoofing, as defined above.

Similarly, authorization is a necessary requirement in order to ensure users have rights to access certain data. Also, what kind of access they have rights to, such as if they can just read from the data or also write to it. Authorization

would prevent the aforementioned elevation of privilege attacks.

Tampering with data can be prevented in the KSS by establishing integrity by way of hash functions, message authentication, and digital signatures.

Unwanted information disclosure can be prevented in the KSS by establishing confidentiality. By functionality, confidentiality is coupled with authorization because it involves ensuring that users can only access information that they are allowed to access. This information could be sensitive data passed from a user's KSApp to the KS-OBU, or between the KSMS and the KSApp. Achieving confidentiality can be done by a number of different encryption methods such as message authentication with authentication encryption.

The KSS must be non-repudiable, and this can be done with things like digital time-stamps, signatures, or audit trails on certain actions performed. This would allow us to objectively know when an action was performed, and who did it.

To prevent Denial-of-Service attacks, the KSS needs to have safeguarded availability. To achieve this server-side, firewalls, as well as Intrusion Prevention/ Detection systems must be put in place. On top of that, the KSApp and KS-OBU will both individually need anti-malware software, and bot-detection to prevent DoS attacks on those components.

4. SEPCAR

SePCAR is a proposed free-floating car sharing system by Symeonidis et al. [4] It uses the Keyless Sharing System model, meaning that it shares the established components. This also means that a number of the previously established security requirements are addressed.

In addition to the already established components of the KSS, SePCAR has an additional component called the public ledger. The public ledger acts as a public bulletin board that the KSMS posts data to for the users to retrieve.

4.1 Model Overview

In this solution the Keyless Car Sharing Management System is decentralized because it works as a complex of different servers that together function to generate and distribute access tokens for cars to users, as well as update and revoke them. The servers each have a part of the car key that they retrieve. Furthermore, each server has its own database independent of the others and secret sharing is utilized for anything that is shared with the KSMS.

When the booking details are converted to secret shares and sent to the KSMS, the servers work together to jointly encrypt the booking details of a given car-share transaction. These encrypted shares can then be posted to the public ledger so that they are kept confidential to any third parties, while allowing the consumer user to safely retrieve the booking details from the ledger. These booking details are what will be used to access the car, along with a digital certificate identifying the consumer user.

Furthermore, SePCAR uses a public-key infrastructure where each component possesses a public and private key. There are also a number of symmetric keys in use.

4.2 SePCAR Functions

SePCAR uses a number of functions that will be utilized during the car sharing process.

- **Open()** is the function that reconstructs a piece of data that has been converted to secret shares that can be called once the caller is in possession of all of the shares.
- **Query_an()** is the function used by a user in order to retrieve data posted to the public ledger. It is run over a secure, anonymous communication channel (Tor is specifically mentioned) [4] in order to keep the user calling this query unidentifiable to any third party.

4.3 Car Sharing Overview

This section will cover the necessary steps that take place in the SePCAR system. SePCAR consists of four steps as defined in the paper: *session key generation and data distribution*, *access token generation*, *access token distribution and verification*, and *car access*. In addition, there are prerequisite steps regarding *car key distribution*, and *car booking*. After the four main steps have taken place, *access token update and revocation* occur as needed.

Car key distribution occurs after a car owner first registers their car with the KSMS, and car booking occurs when a pairing between owner and consumer users agree on a location to pick up the car and begin the process.

4.4 Prerequisite Steps

When an owner registers their car within the KSS, an ID representing the car is created, as well as a symmetric key owned by the KS-OBU. Both of these are converted to secret shares and stored within the KSMS.

When an owner and consumer decide to begin the booking process, the booking details are created. The booking details consist of a hash of the digital certificate from the consumer, the pick up location of the car, a list of conditions specifying how the consumer may use the car, access control rights for which the consumer may use the car, and a booking ID [4].

4.5 Session Key Generation

After the booking process has completed, the owner sends a session key generation request as well as the booking ID to the consumer. This can be seen in Figure 2. The consumer then generates two symmetric session keys, denoted K_1, and K_2. K_1 will used by each of the servers to encrypt the access token to ensure only the consumer can access it. K_2 will be used to create a message authentication code to verify that the access token contains the booking details agreed upon during the car booking process [4]. The consumer then transforms these two keys into secret shares for each server so that the servers cannot individually have access to the session keys but can evaluate functions securely in conjunction. Then the two session keys are encrypted with the public keys of each server. After all of this, the consumer sends an acknowledgement along with the booking ID back to the owner.

The owner waits for the consumer's message of acknowledgement. While this happens, the owner signs the booking details with it's private key, the signature denoted σ^o. After this, the owner converts the pairing of σ^o and the booking details into secret shares (the amount determined again by the number of servers in the KSMS) denoted as $[M^{uc}]$ [4].

After the owner receives the message of acknowledgement, they send each server an access-token generation request, along with the encrypted shares of the session keys, and the shares of the signed booking details.

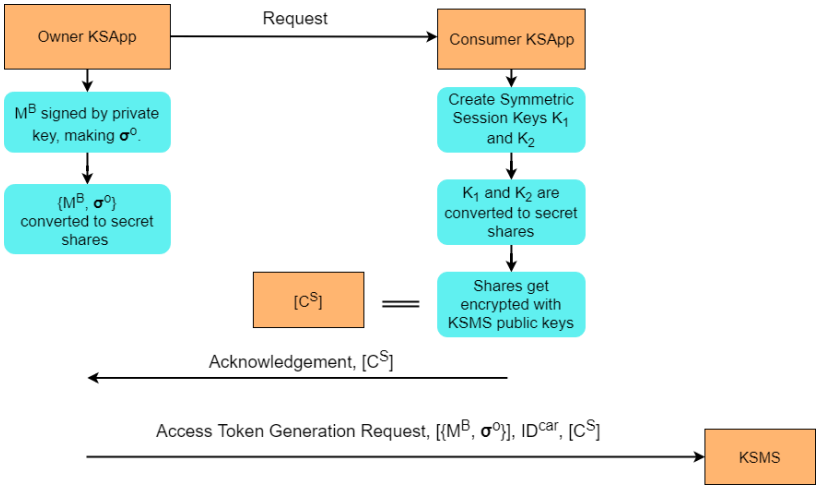

Figure 2: Session key generation and data distribution [4]

4.6 Access Token Generation

In this step, the owner sends an *access key generation request* to the KSMS. Each server uses the booking ID to collectively obtain the shares of the symmetric key of the car. After this, the servers collectively encrypt $[M^{uc}]$ using the obtained symmetric key to generate an access key, denoted $[AT^{car}]$ in shared form across the servers [4].

After this, each server retrieves their shares of the two session keys by decrypting the cipher texts using the private key. Next, the servers encrypt $[AT^{car}]$ and the car ID with the K_1. This generates $[C^{uc}]$. Along with this, the servers collectively perform message authentication. They make a hash of the plaintext booking details alongside K_2, creating the authentication tag, $[C^B]$ (in shared form).

The final step of access token creation is each server sending an *access-token-publication request* to the Public Ledger, along with $[C^B]$, and $[C^{uc}]$.

4.7 Access Token Distribution

Now that the public ledger has received the publication request, an *access-token-publication acknowledgement* is sent to one or more servers within the KSMS, to the owner, and then finally to the consumer. The consumer then retrieves $[C^B]$, and $[C^{uc}]$ with **query_an()** so that the consumer is kept anonymous. After this the consumer can reconstruct the shares with the **open()** function.

With the reconstructed authentication tag, C^B, the consumer is able to verify it's integrity due to the consumer's ownership of the K_2. If the verification is successful, the consumer can use K_1 to decrypt C^{uc}, giving us AT^{car}, and the car ID.

4.8 Car Access

Now the consumer can attempt to access the owner's car. The consumer sends AT^{car}, the car ID, and the unhashed consumer certificate to the KS-OBU via a secure, close range communication channel such as bluetooth [4].

Upon receiving the access components from the consumer, the KS-OBU decrypts AT^{car} back into the digitally signed booking details with the KS-OBU's symmetric key, K^{car}. After this the identities of the owner and consumer can be verified. The owner's identity is verified by decrypting the booking detail's digital signature with the owner's public key, ensuring that it was indeed signed by the owner's private key (and that the contents of the booking details were not altered). Next, the identity of the consumer is verified by comparing the hash of the consumer certificate contained within the booking details to the hash of the consumer certificate obtained directly from the consumer. This way we know the consumer currently accessing the KS-OBU is the same from when the booking details were initially created. If these verification steps are successful, the consumer is allowed access to the car. After the consumer has been allowed access to the owner's car, a timestamp is created and this, paired with the booking details are signed by the KS-OBU and sent to the owner.

4.9 Security Analysis

We will go over how the security requirements laid out in the previous section are met with the SePCAR system.

- Confidentiality in the system is met in three different ways. The booking details are kept confidential due to the fact that they are only shared with the KSMS in a secret shared form. Confidentiality is also established in the access token (AT^{car}), due to the fact that it is generated as secret shares within the KSMS. Furthermore, when it is posted to the public ledger, it is only revealed through its encrypted shared form $[C^{uc}]$. Lastly, confidentiality is established in the KS-OBU's symmetric key (K^{car}) due to the fact that only the car's KS-OBU has access to it, and the KSMS only learns it through distributed secret shares.

- Non-repudiation is established with the access token at two different points. For the AT^{car}'s origin being when the booking details are signed by the owner's private key, this allows both the consumer and KS-OBU to verify this origin with copies of the owner's public key. The delivery of this access token is also made non-repudiable through the message sent from the KS-OBU back to the owner after the consumer is granted access to the car. This allows the owner to receive confirmation from the KS-OBU that the consumer was able to get into and use the car.

- Lastly, integrity is established through the owner's signing of the booking details with their private key. This allows the KS-OBU to verify that these booking details both came from the owner, and haven't been altered in any way.

5. CENTRALIZED APPROACH

In this section we will be going over an additional proposed free-floating car sharing system proposed by Alexandria Dmitrienko and Christian Plappert [2]. Unlike SePCAR this system does not follow the Keyless Sharing System model, but rather has its own established model and proposed security requirements. On top of that, rather than its backend consisting of a decentralized network of servers, there is instead a centralized, trusted car sharing provider that stores and has explicit knowledge of user information.

In addition, there is only a single user considered in this model. This user would be the equivalent of the consumer user from the SePCAR model, and the car they are accessing already exists within the system.

Lastly, prominent to this system is the concept of two-factor authentication when the user is proving their identity to the on-board unit of the car.

5.1 Main differences from SePCAR

5.1.1 Two Factor Authentication

Two factor authentication is achieved by requiring two different authentication factors at the time of car access, both of which are created at different times in order to make it more difficult to compromise both.

The first authentication factor is a key created when the user registers their account with the car sharing system. The second is an access token created by similar means to the SePCAR system. When the user attempts to access the car, they have to prove possession of both of these components that have been created at completely different times [2]. Furthermore, rather than all verification of the user happening on the car's on-board unit, both of these components are instead independently verified on different execution environments. This is so these verification steps are done in isolation, further increasing the difficulty of compromising the car access steps [2].

5.1.2 Centralized Car Sharing Provider

In this system the car sharing provider actively stores and has explicit knowledge of information exchanged during the car sharing process. For instance, the provider stores and gains knowledge of the access policy (availability, location, intended travel distance) in order to determine if the given car can meet these requirements [2].

6. CONCLUSION

In this paper we went over the concept of a Keyless Sharing System, and its various components. From there we covered a threat model of this system in order to identify the necessary security requirements. With an established high-level overview of the system and the requirements in place, we went over two different car-sharing solutions that used the KSS model in order to propose real-world implementations, and how they approach the security requirements.

Acknowledgement

Thank you to Elena Machkasova for the helpful feedback and direction provided towards completing this paper. In addition, thank you to Shawn Seymour for taking the time to review my paper and offer detailed advice. Cryptography and security in general were concepts I had not deeply explored before. All the help I received assisted immensely in tackling the concepts covered in this paper, and ultimately with getting through my senior seminar successfully.

7. REFERENCES

[1] Carsharing market to witness a massive 34% growth over 2016-2024 | markets insider, Jul 2017.

[2] A. Dmitrienko and C. Plappert. Secure free-floating car sharing for offline cars. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, pages 349–360, New York, NY, USA, 2017. ACM.

[3] S. Ray. Cryptographic hashing, Nov 2017.

[4] I. Symeonidis, A. Aly, M. A. Mustafa, B. Mennink, S. Dhooghe, and B. Preneel. Sepcar: A secure and privacy-enhancing protocol for car access provision. In S. N. Foley, D. Gollmann, and E. Snekkenes, editors, *Computer Security – ESORICS 2017*, pages 475–493, Cham, 2017. Springer International Publishing.

[5] P. Symeonidis, Mustafa. Keyless car sharing system: A security and privacy analysis. In *2016 IEEE International Smart Cities Conference*, ISC2. IEEE, 2016.

[6] B. Tournier. Free floating vs. stationary vs. p2p: Car-sharing technology providers open the door to new options.

[7] Wikipedia contributors. Secret sharing — Wikipedia, the free encyclopedia, 2019. [Online; accessed 12-April-2019].