ANTARES - ANonymous Transfer of vehicle Access Rights from External cloud Services

Adriana Berdich, Alfred Anistoroaei, Bogdan Groza, Horatiu Gurban, Stefan Murvay and Daniel Iercan
Faculty of Automatics and Computers, Politehnica University of Timisoara, Romania
{adriana.berdich, bogdan.groza, horatiu.gurban, pal-stefan.murvay, daniel.iercan}@aut.upt.ro, anistoroaeialfred@gmail.com

***Abstract*—As car sharing becomes an increasingly common task, mediating user access rights from external servers comes with threats regarding user's privacy. Clearly, users can be tracked by service mediators, e.g., cloud providers, that manage vehicle fleets, etc. In this work we design and test a simple solution based on oblivious transfer, a well-known and secure cryptographic block, that allows to preserve user's privacy when gaining access to the vehicle. We test the feasibility of deploying such a solution on Android capable smartphones but also account for potential in-vehicle components, e.g., car head units, that may be soon put to such tasks. We use Microsoft Azure as cloud service provider and deploy a Java implementation, based on the Bouncy Castle cryptographic library, on the server side. Our experimental results show that Android based units are capable of handling the required cryptographic operations and the implementation of the employed protocol can be done by existing open-source support.**

***Index Terms*—car-sharing, oblivious transfer, privacy**

I. Introduction and motivation

Traditional car keys are not really able to perform more demanding cryptographic functionalities, e.g., public-key operations, or more complex protocols such as oblivious transfer. In general, traditional car keys are limited to performing basic symmetric cryptographic primitives, e.g., encryption, that are required for challenge-response protocols. Fortunately, the use of smartphones may help in this respect since they are equipped with modern processors that can easily run demanding tasks. Indeed, the use of smartphones as car keys has been suggested in numerous recent works, e.g., [5], [19]. Even earlier than that, the use of smartphones as keys for home and office buildings was explored in [4] and [3].

Recently, multiple lines of work have accounted for various car access systems with advanced functionalities, many of them involving smartphones and the cloud. For example, cloud-to-vehicle communication is discussed in [2] and [17] while the authors of [11] propose the use of the cloud as an infrastructure to control car usage. Car sharing via Android applications is also discussed in [6]. Rights sharing systems over car functionalities are explored in [12] and [9]. Even the use of secure multiparty protocols has been suggested in [15]. Smart-contracts over the Ethereum network for car sharing are proposed in [1]. The work in [8] proposes a more complex role-based car access control system.

This work was supported by a grant of Ministry of Research and Innovation, CNCS-UEFISCDI, project number PN-III-P1-1.1-TE-2016-1317, within PNCDI III (2018-2020).

In the past, there were many reported attacks on car keys, e.g., [7], [16], [18], [20], but these are of little concern to us since most of these attacks come from the absence of proper cryptography, e.g., the use of poor random number generators (RNGs), or poorly designed cryptosystems, e.g., HITAG. In contrast to these, our solution relies on standardized, properly designed cryptographic functions that are available in the Android platform via open-source implementations, e.g., the Bouncy Castle and Spongy Castle cryptographic libraries.

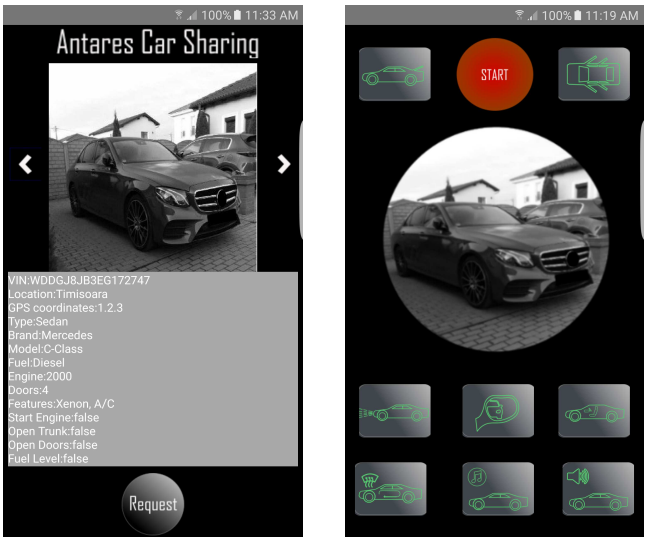

Fig. 1. ANTARES: interfaces for car selection (left) and control (right)

Application and goals. Figure 1 shows the interfaces of the Antares car-sharing app that we designed for Android. The interface allows the users to select, from the available cars, the one that better suits their needs based on specific characteristics or location and perform a rights-request operation following the payment procedure. After a successful procurement of rights over the car, a new application interface can be used for controlling specific functionalities of the car. The retrieval of the access rights to a specific car is done via an oblivious-transfer protocol which will be described in the following section. This allows the selection of a specific car to be done in an anonymous manner. The payment procedure is out-of-scope for the current work. While the payment can be traced (if based on classical payment systems, e.g., VISA) the car that is selected by the user will still remain anonymous due to the oblivious-transfer protocol. Subsequently, each car

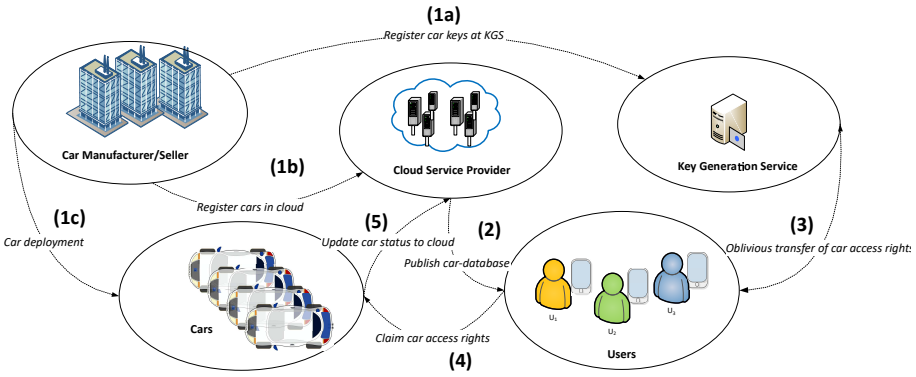

Fig. 2. Addressed scenario in ANTARES: rights procurement from the cloud in an oblivious manner

reports its status in the cloud. If the cloud service provider colludes with the key generation service, some privacy leakage will occur since changes in the status of cars can be linked to credentials that users provide to the key generation service. In this case, the anonymity of the users is enforced as long as multiple cars are rented during the same time-frame. For this purpose, the user application may provide a queuing delay to ensure that sufficient rentals occur at the same time.

II. Protocol design

In this section we discuss the addressed scenario followed by the cryptographic building blocks and our protocol design.

A. Addressed scenario

In Figure 2 we depict the addressed scenario. For simplicity of the exposition, a more formal description of the protocol is given only for the oblivious-transfer (OT) component in the forthcoming section. This is the only challenging protocol procedure and we need to establish precise timing requirements for it. The rest of the protocol procedures involve only basic cryptographic operations which are of little concern to us, thus, we keep the presentation that follows less formal.

We assume that car manufacturers or sellers are responsible with the deployment of vehicles in step 1. For this, in step (1a) the car fleet is registered to a key generation service KGS and in step (1b) to the cloud service provider CSP. The CSP receives all the informations on the car, including technical specifications, pictures, etc. that are displayed in the car sharing application and stores these in a cloud database. The KGS receives only a vehicle identification number which points to a specific vehicle (this can be the VIN number which is unique to each vehicle). The cars are deployed in step (1c) and each car has installed in it the public keys of the trusted KGS. We choose to separate between the cloud service provider CSP and the key generation service KGS since such decentralization precludes the cloud service provider from gaining access to the car and also the KGS from gaining more information on the cars, e.g., their location. If required by implementation constraints, the CSP and KGS can run on the same entity. Even if the CSP colludes with the KGS, the oblivious-transfer protocol prevents them from learning which car was rented by a user U.

Subsequently, in step (2) the user U retrieves the car database from the cloud. We omit details on what the database contains but as can be easily seen in the application interface, the users are informed on the technical specifications of the car. Upon selecting the car they desire, the users run an oblivious transfer protocol with the KGS in step (3). Step (3) is omitted for the moment since we describe it in detail in the next subsection. Having retrieved the access credentials, i.e., $\mathsf{Sig}_{\mathsf{KGS}}(attributes, \mathsf{Car}_{\mathsf{id}})$ in step (3), the users access the car in step (4). This is simply done by sending these credentials over the Internet to the car infotainment unit. The attributes attr contain the user rights over the car and the lifetime of the acquisition. Upon successful access, the car updates its status to the cloud in step (5) by simply switching its availability flag. To avoid the same car being rented at the same time by two distinct users (since the KGS does not know if the keys that he released are for available cars) it is the role of the CSP to maintain the updated list of available cars. To avoid inconsistencies due to concurrent rentals of the same car, the CSP has to lock the car for the duration of the oblivious-transfer protocol between the user and the KGS.

B. The oblivious transfer step

Oblivious transfer stays at the core of our protocol design. This building block is required such that the KGS does not learn to which of the cars the user U has gained access. Oblivious transfer was introduced by Rabin in [14]. In our protocol implementation we stay to the more recent efficient oblivious transfer protocol introduced in [13] following its description from [10]. This protocol has its security based on the Decisional Diffie-Hellman problem and we implement it on elliptic curves that provide a more compact representation. This 1-out-of-2 protocol can be easily extended to a generic case of 1-out-of-n oblivious transfer.

We formally present protocol details in Algorithm 1, which is a translation of the Naor-Pinkas protocol [13] described in [10] to the syntax of elliptical curves. We assume the existence of publicly-known point P of an elliptical curve, i.e., $P \in E(\mathbb{F}_p)$, with order q (subsequently, all constants

that are used for point multiplication are part of $\mathbb{F}_q^*$, i.e., the set of invertible elements modulo q). The algorithm presents the interaction between user U and the key generation service KGS. We denote the set of cars as $\mathsf{Car}_i, i = 1, l$ (where l is the maximum number of cars accepted by the protocol setup) and let $i_n, i_{n-1}, ..., i_1, i_0$ denote the binary expansion of their index i. Here the bit-length of the index follows as $n = \lceil \log_2 l \rceil$. For each car a master key KM is derived which is used to encrypt the rights that are shared for the car. The master key is derived as follows: $\mathsf{KM}_i = \mathcal{KD}(\mathsf{KM}, \mathsf{K}_{\mathsf{left}})$ *iff* $i_j = 0$ else $\mathcal{KD}(\mathsf{KM}, \mathsf{K}_{\mathsf{right}})$ *iff* $i_j = 1$. Here $i = 1..l$ and $j = 1..n$. This key is used to encrypt the rights for the car which are signed by the KGS as $E_{\mathsf{KM}_i}\{\mathsf{Sig}_{\mathsf{KGS}}(\mathsf{attr}, \mathsf{Car}_{\mathsf{id}})\}$. Note that the KGS creates such a certificate for each car and in the following oblivious-transfer protocol it remains unknown for the KGS which of the keys was retrieved by the user U. The binary expansion of the car index, used for key derivation, is subsequently used in the oblivious transfer protocol. That is, the protocol is run for each bit of the index and user U retrieves at each step $j = 0..n$ from KGS either $\mathsf{K}_{\mathsf{left}}$ if $i_j = 0$ or $\mathsf{K}_{\mathsf{right}}$ if $i_j = 1$. The master key KM is derived at each step by adding the newly extracted key value, i.e., $\mathsf{KM} \leftarrow \mathcal{KD}(\mathsf{KM}, k)$. Inside the *for* loop there is an 1-out-of-2 oblivious transfer protocol from [13] which we refined for the syntax of elliptical curves. We prefer elliptical curves instead of integer group Z_p due to the more compact size.

Algorithm 1: Oblivious transfer based on Naor-Pinkas protocol [13], [10] in the syntax of elliptic curves

Result: Key for car $i = i_n i_{n-1} ... i_0$
initialization;
for $l = 0$ *to* n **do**
  1.1 U: $a, b, r \leftarrow_R \mathbb{F}_q^*$;
  if $i_l = 0$ **then**
    1.2 U: $\mathsf{send}(aP, bP, abP, rP)$;
  else
    1.2 U: $\mathsf{send}(aP, bP, rP, abP)$;
  end
  2.1 KGS: $\mathsf{receive}(X, Y, Z, T)$;
  2.2 KGS: $u_0, u_1, v_0, v_1 \leftarrow_R \mathbb{F}_q^*$;
  2.3 KGS: $w_0 \leftarrow u_0 X + v_0 P$, $w_1 \leftarrow u_1 X + v_1 P$;
  2.4 KGS: $k_0 \leftarrow u_0 Z + v_0 Y$, $k_1 \leftarrow u_1 T + v_1 Y$;
  2.5 KGS: $c_0 \leftarrow k_0 \oplus \mathsf{K}_{\mathsf{left}}$, $c_1 \leftarrow k_1 \oplus \mathsf{K}_{\mathsf{right}}$;
  2.6 KGS: $\mathsf{send}(w_0, c_0, w_1, c_1)$;
  3.1 U: $\mathsf{receive}(w_0, c_0, w_1, c_1)$;
  if $i_l = 0$ **then**
    3.2 U: $k \leftarrow bw_0$, $\mathsf{K}_{\mathsf{left}} \leftarrow k_0 \oplus c_0$;
  else
    3.2 U: $k \leftarrow bw_1$, $\mathsf{K}_{\mathsf{right}} \leftarrow k_1 \oplus c_1$;
  end
  3.3 U: $\mathsf{KM} \leftarrow \mathcal{KD}(\mathsf{KM}, k)$;
end

III. Experiments and results

This section presents results on rights procurement procedures and access to the in-vehicle components in the proposed setup.

A. Rights procurement via the cloud

For the cloud-based implementation, we used the Microsoft Azure Cloud Services. The cars are stored in a SQL database in the Cloud. To view the available cars from the SQL database, we developed an Android application. We use a Web App Service for the mobile application on Android and specify a connection string to the SQL database. .NET was used to develop the server side, i.e., the back-end, of the mobile application. The Web App Service is a MVC (Model-View-Controller) implemented in .NET which also generates the structure of the table and updates it. When deploying the back-end application in Azure, a table containing car records is automatically generated based on the MVC in the SQL database. The connection between the Android application and the SQL database was done based on the URL of our Web App Service. The Android application that runs on user smartphones allows users to view available cars and to reserve them. When a car is booked the table is updated by another Android application that runs on every car head unit (an ERISIN infotainment unit). Since each car must update only its own data, we use the Row Level Security functionality of SQL which enables access from the car head unit only to the specific line dedicated to it in the database. The key extraction via the oblivious-transfer protocol is implemented in Java with the help of the Spongy Castle[1] (for the Android devices) and Bouncy Castle[2] (for the Java server side on Azure) cryptographic libraries.

Computational times for the steps of the 1-out-of-2 oblivious transfer on several curves (160, 192 and 256 bit) are shown in Table I for Android devices and Table II for servers. The computational time is only up to around 20 milliseconds for servers and up to a few hundred milliseconds for smartphones. To extend to a 1-out-of-n oblivious transfer, these should be multiplied by $\lceil \log_2(n) \rceil$ which leads only to a modest growth. Figure 3 gives a graphical depiction for the computational time of the protocol steps on smartphones and the head unit.

Next we tested the performance in case of running the entire protocol between a smartphone and the server and how the server can handle clients on multiple threads. Figure 4 shows the computation time when running 1-60 threads for client computation on the Azure virtual machine. One of our Azure subscription had only 1 vCPU, the performance will obviously increase with the number of cores. This can be easily seen for the second virtual machine which has 2 vCPU. Finally, Figure 5 shows the predicted variation of computational time with the number of clients and cores. Even for a poor 1 vCPU server, 200 clients can be served by the oblivious transfer protocol in a matter of minutes (for a large $n = 20$). Table III shows comparative results for 1 and 20 client threads on the two types of virtual machines. Since the second one has two vCPUs it is expected that the runtime is almost half for multiple client threads. There are no improvements for a single thread since the server cryptographic operations are not parallelized, a few

[1] https://rtyley.github.io/spongycastle/
[2] https://www.bouncycastle.org/

TABLE I
COMPUTATIONAL TIME FOR THE 1-OUT-OF-2 OT ON ANDROID DEVICES

	secp160r1		secp192r1		secp256r1	
Steps	1.1-1.2	3.2	1.1-1.2	3.2	1.1-1.2	3.2
Samsung J5	$31ms$	$5ms$	$43ms$	$8ms$	$66ms$	$11ms$
Samsung S7	$85ms$	$7ms$	$81ms$	$7ms$	$103ms$	$9ms$
Allview	$77ms$	$11ms$	$117ms$	$12ms$	$159ms$	$31ms$
One Plus 7 Pro	$6ms$	$0.8ms$	$9ms$	$1.2ms$	$9ms$	$1.2ms$
ERISIN	$55ms$	$10ms$	$70ms$	$10ms$	$116ms$	$19ms$

TABLE II
COMPUTATIONAL TIME FOR THE 1-OUT-OF-2 OT ON SERVER

	secp160r1	secp192r1	secp256r1
Steps	2.1-2.6	2.1-2.6	2.1-2.6
Azure VM Standard B1s	$19ms$	$21ms$	$20ms$
Azure VM Standard F2s_v2	$16ms$	$16ms$	$16ms$

hundred milliseconds are not restrictive in any sense however. Finally, Table IV gives the runtime of the protocol when run between each of the smartphones and the Azure server. The runtime is depicted for a maximum $n = 20$ which allows oblivious selection from more than 1 million distinct keys, i.e., 2^{20}. We choose this value for n only as an upper bound, in a realistic scenario the value of n would be much smaller, e.g., $n = 10$ would allow the user to anonymously select from more than one thousand cars.

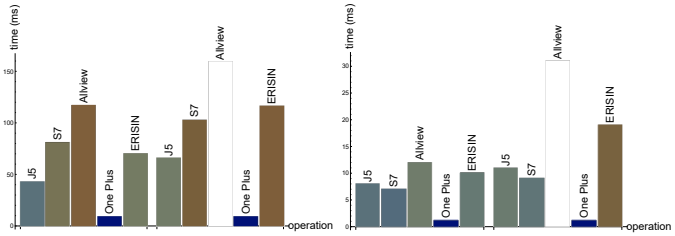

Fig. 3. Computational time for the first (left) and second protocol steps (right) of the 1-out-of-2 oblivious transfer protocol

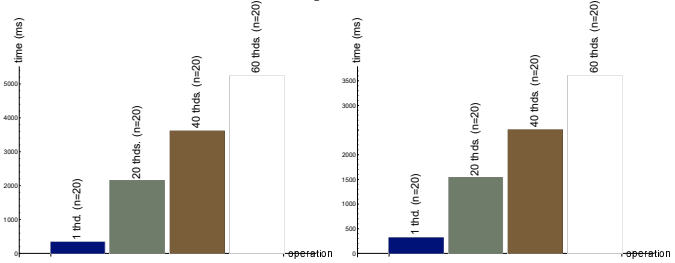

Fig. 4. Computational time for 1-60 client threads on an B1 Azure machine (left) and on an F2 Azure machine (right) for $n = 20$

B. Access to in-vehicle modules in the experimental setup

For a more realistic scenario, we further used the experimental setup of the CSEAMAN Project, i.e., a project related to the security of in-vehicle systems, to test remote connectvity to in-vehicle units. The setup consists of several automotive off-the-shelf parts and development boards equipped with automotive

TABLE III
COMPUTATIONAL TIME FOR THE 1-OUT-OF-N OT PROTOCOL FOR SERVER IN CASE OF MULTITHREADING (SECP256R1 CURVE)

Client threads	20	20	40	60
n	1	20	20	20
Azure VM Standard B1s	$325ms$	$2132ms$	$3618ms$	$5239ms$
Azure VM Standard F2s_v2	$318ms$	$1528ms$	$2506ms$	$3600ms$

TABLE IV
PROTOCOL RUNTIME BETWEEN PHONES AND AZURE SERVER

	secp160r1		secp192r1		secp256r1	
n	J5	S7	J5	S7	J5	S7
4	$476ms$	$554ms$	$501ms$	$528ms$	$594ms$	$782ms$
8	$971ms$	$1221ms$	$1178ms$	$1342ms$	$1216ms$	$2007ms$
10	$1176ms$	$1472ms$	$1246ms$	$1598ms$	$1413ms$	$2405ms$
12	$1515ms$	$1807ms$	$1489ms$	$1861ms$	$1639ms$	$2877ms$
16	$2041ms$	$2343ms$	$2056ms$	$2545ms$	$2185ms$	$3864ms$
20	$2525ms$	$3032ms$	$2583ms$	$3043ms$	$2810ms$	$4965ms$

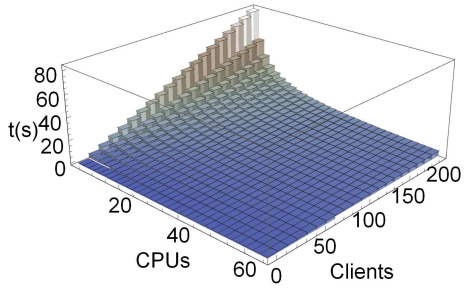

Fig. 5. Expected runtime for the 1-out-of-n protocol with the number of clients and CPUs in the cloud (n=20)

grade microcontrollers. The experimental setup interconnects those items by using LIN (Local Interconnect Network), FlexRay, low-speed and high-speed CAN (Controller Area Network) buses. ECUs inside vehicles are clustered into several networks based on their functionality. Primarily they are grouped in the following domains: Body, Infotainment and Telematics, Chassis and Powertrain. The experimental setup was designed to use the same grouping of components.

The main experimental setup components are an acceleration pedal on high-speed CAN, an instrument cluster on low-speed CAN, a RLS-Rain Light Sensor on LIN and development boards using the NXP S12DG128 and MC9S12XF512 controllers. All the networks are interconnected by employing a gateway with intrusion detection functionality. This is implemented by using a NXP S12DG128 microcontroller. Beside the gateway functionality this also implements some BCM (Body Control Module) functions to control the low beam headlights, rear lights, front and rear turn signal lights.

We encountered problems in writing data from the infotainment unit that we used directly to the CAN bus (despite the fact that the unit is capable of CAN communication). As a

temporary patch, we used a Raspberry Pi 3 development board as a Wi-Fi bridge between the Infotainment unit and the CAN bus. We chose this board due to its ability to run Linux which allows us to use a multitude of available libraries. Because the Raspberry Pi 3 microprocessor, the ARM Cortex A53, does not include an CAN controller an external SPI connected board with Microchip MCP2515 CAN controller and NXP TJA1050 CAN transceiver was added to the setup. The CAN communication is supported in LINUX by using SocketCAN API which provides also a set of drivers for the CAN interface. In out setup Raspberry is connected to the BCM using a high speed CAN network.

Two applications were developed. The first application is used to run a simulation environment providing the information needed by the devices to bypass the startup phase and to enter into normal functioning mode. The application is used to monitor and control different experimental stand functionalities. It provides an interface for enabling/disabling the instrument cluster warning lamps and indicators and for modifying the values displayed on the gauges. The simulation environment and the control and monitoring panels developed for the CSEAMAN project where implemented by using Vector's CANoe software. This new implementation was desired because we wanted to have an alternative where no commercial software is used allowing us to continue the development of the experimental setup without the need of commercial software. The second application allows remote monitoring and control of basic functionalities. It allows the user to be informed about the status of following warning lights and indicators from the cluster: ABS/ESP system fault, parking brake, low fuel, battery/alternator warning, low tire pressure, front and rear lights. In case when the car is turned off, the BCM does not respond to requests from Raspberry so the last acquired values will be sent to the user. If the vehicle is stationary the mobile app can be used to control the low beam headlights, rear lights, front and rear turn signal lights, and the hazard lights, etc.

IV. Conclusion

Our work explores a solution for gaining access to vehicles in an oblivious manner. In this way the key-generation service remains un-aware on the specific car for which it released access to the user. The experiments prove that the required cryptographic functionalities can be easily executed by modern Android devices, i.e., smartphones and vehicle head units, as computational requirements are in the order of hundred milliseconds for the oblivious-transfer block. Such solutions are promising for assuring user's privacy in car sharing scenarios that are very common nowadays. Even with the modest performance of our current Azure subscription, limited to one vCPU, the server side could easily handle connections from several clients each second. The current work was intended only as proof-of-concept to verify that more demanding cryptographic blocks, e.g., oblivious transfer, are feasible for this scenario, a more complete implementation of the protocol that conforms to real-world needs, may be subject to future work.

References

[1] M. Akash, I. Symeonidis, M. A Mustafa, B. Preneel, and R. Zhang. Sc2share: smart contract for secure car sharing. In *International Conference on Information Systems Security and Privacy (ICISSP)*, 2019.

[2] G. Alli, L. Baresi, A. Bianchessi, G. Cugola, A. Margara, A. Morzenti, C. Ongini, E. Panigati, M. Rossi, S. Rotondi, et al. Green move: towards next generation sustainable smartphone-based vehicle sharing. In *Sustainable Internet and ICT for Sustainability (SustainIT), 2012*, pages 1–5. IEEE, 2012.

[3] L. Bauer, L. Cranor, R. W Reeder, M. Reiter, and K. Vaniea. Comparing access-control technologies: A study of keys and smartphones. 2007.

[4] L. Bauer, L. F. Cranor, M. K. Reiter, and K. Vaniea. Lessons learned from the deployment of a smartphone-based access-control system. In *SOUPS*, 2007.

[5] C. Busold, A. Taha, C. Wachsmann, A. Dmitrienko, H. Seudié, M. Sobhani, and A.-R. Sadeghi. Smart keys for cyber-cars: secure smartphone-based nfc-enabled car immobilizer. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 233–242. ACM, 2013.

[6] 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*, pages 349–360. ACM, 2017.

[7] A. Francillon, B. Danev, and S. Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In *NDSS*, 2011.

[8] B. Groza, T. Andreica, A. Berdich, P.-S. Murvay, and H. Gurban. Prestvo: Privacy enabled smartphone-based access to vehicle on-board units. *arXiv preprint arXiv:1911.05144*, 2019.

[9] B. Groza, T. Andreica, and P.-S. Murvay. Designing wireless automotive keys with rights sharing capabilities on the msp430 microcontroller. In *Proc. 3rd Int. Conf. Veh. Technol. Intell. Trans. Syst.*, pages 173–180.

[10] C. Hazay and Y. Lindell. *Efficient secure two-party protocols: Techniques and constructions*. Springer Science & Business Media, 2010.

[11] Y.-S. Huang and C.-H. Lung. Device with identity verification—apply in car driving as an example. In *2018 IEEE International Conference on Applied System Invention (ICASI)*, pages 243–246. IEEE, 2018.

[12] T. Kasper, A. Kühn, D. Oswald, C. Zenger, and C. Paar. Rights management with nfc smartphones and electronic id cards: A proof of concept for modern car sharing. In *Radio Frequency Identification*, pages 34–53. Springer Berlin Heidelberg, 2013.

[13] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 448–457. Society for Industrial and Applied Mathematics, 2001.

[14] M. O. Rabin. How to exchange secrets with oblivious transfer. 1981.

[15] 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 *European Symposium on Research in Computer Security*, pages 475–493. Springer, 2017.

[16] S. Tillich and M. Wójcik. Security analysis of an open car immobilizer protocol stack. In *Trusted Systems*, pages 83–94. Springer, 2012.

[17] J. Timpner, D. Schürmann, and L. Wolf. Secure smartphone-based registration and key deployment for vehicle-to-cloud communications. In *Proceedings of the 2013 ACM workshop on Security, privacy & dependability for cyber vehicles*, pages 31–36. ACM, 2013.

[18] R. Verdult, F. D. Garcia, and J. Balasch. Gone in 360 seconds: Hijacking with hitag2. In *Proceedings of the 21st USENIX conference on Security symposium*, pages 37–37. USENIX Association, 2012.

[19] Z. Wei, Y. Yanjiang, Y. Wu, J. Weng, and R. H. Deng. Hibs-ksharing: Hierarchical identity-based signature key sharing for automotive. *IEEE Access*, 5:16314–16323, 2017.

[20] J. Wetzels. Broken keys to the kingdom: Security and privacy aspects of rfid-based car keys. *arXiv preprint arXiv:1405.7424*, 2014.