

CarINA - Car sharing with Identity based Access control re-enforced by TPM

Bogdan Groza¹, Lucian Popa¹, and Pal-Stefan Murvay¹

Faculty of Automatics and Computers,
Politehnica University of Timisoara, Romania,
{bogdan.groza, lucian.popa, pal-stefan.murvay}@aut.upt.ro

Abstract. Car sharing and car access control from mobile devices is an increasingly relevant topic. While numerous proposals started to appear, practical deployments ask for simple solutions, that are easy to implement and yet secure. In this work we explore the use of TPM 2.0 functionalities along with identity-based signatures in order to derive a flexible solution for gaining access to a vehicle. While TPM 2.0 specifications do not have support for identity-based primitives we can easily bootstrap identity-based private keys for Shamir's signature scheme from regular RSA functionalities of TPM 2.0. In this way, key distribution becomes more secure as it is re-enforced by hardware and the rest of the functionalities can be carried from software implementations on mobile phones and in-vehicle controllers. We test the feasibility of the approach on modern Android devices and in-vehicle controllers as well as with a recent TPM circuit from Infineon.

1 Introduction and Motivation

Despite their simplicity and well established foundational concepts, i.e., authentication protocol, traditional car keys tell a long shameful story about insecurity, e.g., [8], [18], [20], [22]. The use of smartphones as car keys has been proposed in numerous works. For example [3] proposes a complex platform for car access and rights delegation and uses a secure microSD smart-card to further increase the security level. Identity-based cryptography for car sharing has been also proposed in [21]. Some works go even further by proposing secure multiparty protocols [17] but these may be too computational intensive for the current infrastructure.

Our goal in this work is to deploy a simple solution that takes advantage of identity-based primitives and also of the increased security provided by the use of TPM (Trusted Platform Module) devices. Naturally, we require for the solution to be deployed on modern smartphones and in-vehicle units. The use of modern smartphones comes with many advantages since numerous functionalities can be implemented, e.g., car sharing and car localization, etc., while the flexibility of Java support opens road for numerous cryptographic primitives. In particular, identity-based cryptography has the advantage that it does not require storing public-key certificates. Handling certificates in particular may be uneasy on in-vehicle components. In contrast, using principal identities is more easy to handle and nonetheless it offers better anonymity since users can choose pseudonyms that leave no traces inside the car. Of course, due to legal purposes, users

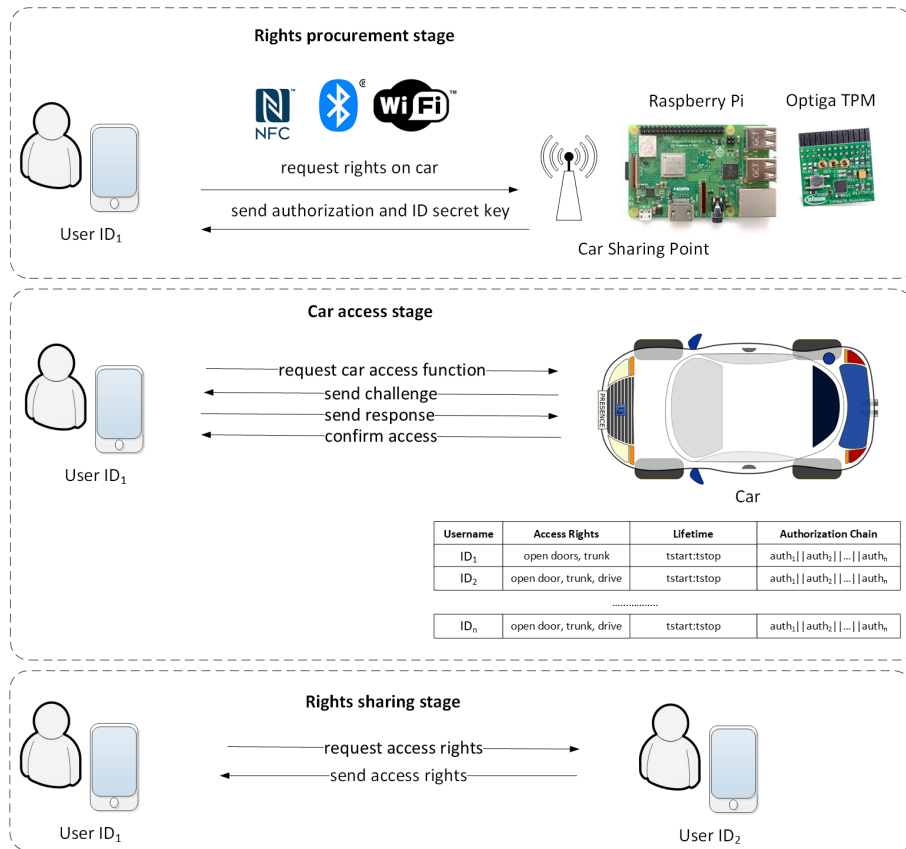


Fig. 1. CARINA: concept and scenarios

should still need to provide proper credentials and prove their right to drive the car at a sharing center. But the privacy of the user should not be exposed inside the vehicle that is rented.

Scenarios. Figure 1 shows a graphical depiction of the scenario that we address. As stated, we opt for identity-based cryptography since it is more intuitive, it protects user's privacy by relying on pseudonyms and avoids storing and sending digital certificates which may be a problem both because it requires bandwidth and due to storage memory constraints. In the depicted scenario, users receive their identity-based private keys from a sharing point via NFC connectivity. The sharing-point is implemented around Raspberry Pi and the TPM circuit to ensure security. The sharing-point may be in an unsupervised place and may be subject to physical access by the adversary, a reason for which the hardware security offered by TPM 2.0 is a significant advantage. NFC is suggested due to its short range which makes it harder to compromise but other interfaces may be used, even very long range such as LTE, provided that they are sufficiently secure. To make credentials spoofing unfeasible, the credentials may be

encrypted and the user transaction key may be sent via an additional channel, e.g., SMS or e-mail. Whenever requesting access to the car a challenge-response protocol is run. The car stores the users identities in a table similar to Unix credential files, e.g., the passwd files, along with the expiry time and the access rights, e.g., open the car or start the engine. Not all users will be granted full rights on the car. For example, a passenger may be allowed to open the car but may not start the engine assuming that he has no driving license. Subsequently, users may further share their rights to other users over Bluetooth, again in a challenge-response fashion. All these actions will be discussed in details in a forthcoming section dedicated to our protocol.

1.1 Related Work

One of the first mentions of using trusted platform modules in the context of automotive systems comes as early as 2004 [2]. In parallel with the development of generic TPM specifications, researchers in the automotive area have proposed the use of a Hardware Security Module (HSM) with TPM-like functionalities to suit the specific requirements of the vehicular environment. Wolf et al. propose the use of a HSM for implementing an automotive digital rights management system [24]. The work in [23] describes the design, implementation and evaluation of an HSM for vehicular environments. The comparison of the three proposed HSM variants with specifications of an industry-proposed vehicular secure hardware, smart cards and TPM 1.2 devices illustrates superior features in all but the light HSM implementation.

The use of TPM functionalities were proposed for various automotive applications especially those in which the vehicle communicates with the outside environment, where existing automotive grade platforms cannot provide adequate performance for software implementations of security solutions. The usage of TPMs for implementing security in Vehicular Ad-Hoc Networks (VANETs) was proposed in [11]. The authors of [9] use a TPM as root of trust for their implementation of car-to-car communication system. Proposals for over-the-air automotive firmware updates involve public key operations which could be efficiently implemented in the vehicle with the use of a TPM or HSM. One such approach which uses an off-the-shelf TPM chip connected to the wireless vehicle unit is proposed in [16].

Another use case is the communication between the vehicle and smart devices. An example is the use of smartphones to delegate usage rights over cars in a car sharing/renting application. Symeonidis et al. [17] propose such a system in which the on-board unit responsible with the verification of usage rights is equipped with a HSM providing secure key storage and support for cryptographic operations.

2 Components

Table 1 provides a summary of the platforms that we used in our work. We discuss more details on the TPM and TriCore controllers next. Figure 2 depicts the setup of our work: an Infineon Optiga TPM connected to a Raspberry Pi.

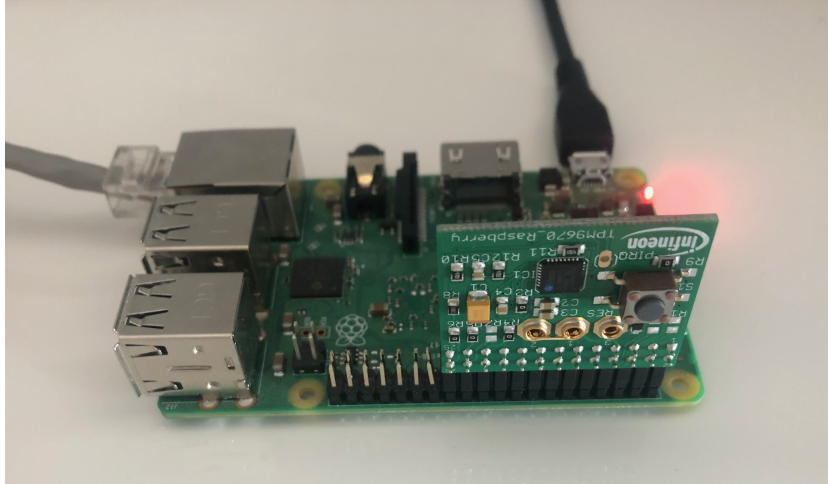


Fig. 2. Experimental setup: the Infineon Optiga TPM connected to a Raspberry Pi

Table 1. Platforms used in our work

Platform	Core	Flash	RAM	Clock	Manufacturer
Optiga TPM	16-bit CPU	6962B	not specified	not specified	Infineon
TriCore TC297	TriCore 1.6P, 32-bit CPU	8MB	728KB	300MHz	Infineon
Raspberry Pi 3B+	Cortex-A53, Quad-Core, 64-bit CPU	32 GB	1GB	1.4GHz	Raspberry Pi Foundation
Samsung Note 8	Exynos 8895, Octa-Core, 2.3GHz Quad + 1.7GHz Quad	128GB	6GB	2.3GHz	Samsung

The trusted platform module (TPM) is a security standard which was defined by the Trusted Computing Group (TCG)¹ and standardized as ISO/IEC 11889:2009 for TPM 1.2 and ISO/IEC 11889:2015 for TPM 2.0. There are various properties of the TPM which makes it an ideal solution for hardware-based security operations such as secure storage of security data (e.g. cryptographic keys), keeping track of the running platform software trust state using platform configuration registers (based on integrity measurements recorded before each software application is executed), generation of symmetric keys or asymmetric key-pairs based on a unique, externally inaccessible, endorsement key and generation of random numbers using a TRNG.

For our experiments, we chose to use the OPTIGA SLB 9670XQ2.0 TPM from Infineon which is compliant to the specifications of TPM 2.0. We used the OPTIGA TPM evaluation board mounted on a Raspberry Pi 3B+ for implementing the protocol building blocks. The OPTIGA TPM datasheet, its parametrics and other technical doc-

¹ <https://trustedcomputinggroup.org>

uments are given by Infineon as public information [14]. Based on the OPTIGA TPM parametrics, we determined that it can execute, by requests over the SPI interface, the following commands: i) generate randomness, ii) asymmetric encryption/decryption on a given input using a loaded key (i.e., RSA-2048 encryption), iii) asymmetric signing/verification given the input and using a loaded key (i.e., ECDSA-256 signature), iv) symmetric signing/verification of a given input and using a loaded key (i.e., HMAC - Hash-based Message Authentication Code), v) computing a hash function (i.e., SHA-256) on a given input. Also, according to the TCG requirements for TPM 2.0 and the OPTIGA TPM datasheet we determined that the chip has the following storage characteristics: i) 6962 bytes of non-volatile memory, ii) 1420 bytes I/O buffer, iii) 1024 bytes for command/response parameters, iv) 768 bytes for non-volatile read/write operations, v) can handle up to 7 objects loaded in the non-volatile memory area, vi) can handle up to 3 objects loaded in the volatile memory area.

Given the open nature of the TPM 2.0 library specification there exist open-source libraries containing the implementation of TPM functions (e.g., <https://github.com/Infineon/eltt2>, <https://github.com/tpm2-software>). These libraries can be built and executed on embedded devices in order to send any of the presented commands and receive the requested data from a TPM device. Additionally to the communication with the real trusted platform module there is also the possibility to use a tpm2-simulator <https://sourceforge.net/projects/ibmswtpm2/> on a Linux machine with the mentioned open-source libraries in order to send/receive commands. This simulator can be used for testing and development as it emulates a hardware TPM.

3 Protocol

In this section we first discuss the building blocks behind our protocol proposal, then we give precise details on the protocol.

3.1 Cryptographic Building Blocks

We rely on standardized cryptographic primitives for encrypting, i.e., AES, and message authentication codes, i.e., HMAC. In addition to these, we use Shamir's identity-based signature [15] which can be easily described in what follows:

1. $\text{Setup}(k)$ is the key setup algorithm that generates the master secret key msk and the public key pk . The Setup algorithms, generates two random primes p, q , each having k bits in length, computes $n = pq$, $\phi(n) = (p - 1)(q - 1)$, selects random integer $e \in \mathbb{Z}_{\phi(n)}$ s.t. $\text{gcd}(e, \phi(n)) = 1$ and computes $d = e^{-1} \pmod{\phi(n)}$. The master secret key is $\text{msk} = \{n, d\}$ and the public key is $\text{pk} = \{n, e, h\}$. Here h is a hash function that maps the name of a user to an element of $\mathbb{Z}_{\phi(n)}$, i.e., $h : \{0, 1\}^* \rightarrow \mathbb{Z}_{\phi(n)}$.
2. $\text{KeyDer}(\text{msk}, I)$ is the key derivation algorithm that uses the master secret key msk and the identity of the user I to generate his private key by computing $I^d \pmod{n}$. The user secret key is $\text{sk} = \{I^d \pmod{n}, n\}$ (the public key to verify the signatures of this user is the identity of the user I along with system parameters pk).

3. $\text{Sign}(\text{sk}, m)$ is the signature algorithm that takes as input the user's secret key sk and a message m and returns the signature σ . For this, the signing algorithm selects a random $r \in Z_n$, computes $t = r^e \bmod n$, the hash of t concatenated with message m denoted as $h = \text{hash}(t||m)$, then $s = I^d r^h \bmod n$. The signature is $\sigma = \{s, t\}$.
4. $\text{Ver}(\text{pk}, I, m, \sigma)$ is the verification algorithm which takes as input the system parameters pk , the identity of the user I , the message m and the signature σ and returns *true* if the signature is correct otherwise it returns \perp . To verify that the signature is correct the algorithm computes s^e and checks if this is equal to $It^h \bmod n$ and returns *true* if so or \perp otherwise.

Identity-based primitives are not supported by the current TPM 2.0 specifications. Though, current standards such as the ISO/IEC 14888-2:2008 support identity-based signatures based on the Guillou-Quisquater scheme [12] for use in embedded devices such as smart-cards [13]. By using regular RSA support from TPM 2.0 we can however easily bootstrap identity-based keys for Shamir's scheme as we discuss in the experimental section. The reasons for choosing Shamir's scheme [15] in favour of Guillou-Quisquater scheme [12] was its simplicity and straight-forward way to derive secret keys from our TPM circuit.

3.2 Protocol Description

The proposed protocol consists of three stages: rights procurement from the car sharing center, the car access and the rights delegation sub-protocols. For brevity we do not include a rights revocation procedure. User rights have an expiry time, if rights need to be revoked sooner than that, then the car sharing entity should be able to maintain a revocation list inside each car which takes priority over the credential of the user. This should be easy to deploy if the cars have Internet connectivity but is out of scope for this work. Protocol procedures are summarized in Figure 3 and we discuss each step in detail next.

The *rights procurement* stage occurs over a secure channel. We assume that this happens at a registration desk or, in case it is an unsupervised selling point, we assume a secure short-range interface such as NFC. If this is unavailable, then the credentials can be encrypted and the encryption key sent by a secondary channel such as SMS or e-mail. We do not insist on this additional procedure. The user registered by ID_{Usr} requests rights on car ID_{Car} from sharing center ID_{Shr} . The rights are encoded in the string func^* and may consist in full rights over the car or maybe just some restricted functions, e.g., opening the trunk in case the user forgets some belongings from a previous sharing. The rights start at current time encoded in time and have a fixed lifetime lifetime . The sharing center will check that current time time does not drift significantly from a real-time clock (drifts in the order of seconds should be acceptable). To grant credentials, the sharing center returns a secret identity-based key $\text{sk}(\text{ID}_1)$ and signs the rights of the user as $s_{\text{Shr}} = \text{Sig}(\text{sk}(\text{Shr}), m_{\text{Usr}})$.

The *car access* stage occurs over an insecure channel, e.g., Bluetooth or WiFi, between the smartphone of the user and some in-vehicle controller (e.g., an embedded

<p>I) Rights procurement (secure channel)</p> <ol style="list-style-type: none"> 1. $\text{Usr} \rightarrow \text{Shr}: m_{\text{Usr}} = \{\text{ID}_1, \text{ID}_{\text{Car}}, \text{ID}_{\text{Shr}}, \text{func}^*, \text{time}, \text{itime}\}$ 2. $\text{Shr} \rightarrow \text{Usr}: s_{\text{Shr}} = \text{Sig}(\text{sk}(\text{Shr}), m_{\text{Usr}})$ 3. Usr sets $\langle \text{authChain} \rangle_{\text{Usr}} = \{m_{\text{Usr}}, s_{\text{Shr}}\}$
<p>II) Car access (insecure channel)</p> <ol style="list-style-type: none"> 1. $\text{Usr} \rightarrow \text{Car}: m'_{\text{Usr}} = \{\text{ID}_{\text{Usr}}, \text{ID}_{\text{Car}}, \text{func}^*, \text{rand}_{\text{Usr}}^{128}\}, \langle \text{authChain} \rangle_{\text{Usr}}$ 2. $\text{Car} \rightarrow \text{Usr}: m'_{\text{Car}} = \{\text{rand}_{\text{Car}}^{128}\}, s'_{\text{Car}} = \text{Sig}(\text{sk}(\text{Car}), m'_{\text{Car}} m'_{\text{Usr}})$ 3. $\text{Usr} \rightarrow \text{Car}: s'_{\text{Usr}} = \text{Sig}(\text{sk}(\text{Usr}), m'_{\text{Usr}} m'_{\text{Car}})$ 4. $\text{Car} \rightarrow \text{Usr}: s''_{\text{Car}} = \text{Sig}(\text{sk}(\text{Car}), s'_{\text{Usr}})$
<p>III) Rights sharing (insecure channel)</p> <ol style="list-style-type: none"> 1. $\text{Usr}_1 \rightarrow \text{Usr}_2: m'_{\text{Usr}_1} = \{\text{ID}_{\text{Usr}_1}, \text{ID}_{\text{Car}}, \text{ID}_{\text{Usr}_2}, \text{func}^*, \text{time}, \text{itime}\},$ $s'_{\text{Usr}_1} = \text{Sig}(\text{sk}(\text{Usr}_1), m'_{\text{Usr}_1})$ 2. $\text{Usr}_2 \rightarrow \text{Usr}_1: m'_{\text{Usr}_2} = \{\text{ID}_{\text{Usr}_2}, \text{ID}_{\text{Car}}, \text{ID}_{\text{Usr}_1}, \text{func}^*, \text{rand}_{\text{Usr}_2}^{128}, \text{time}, \text{itime}\},$ $\langle \text{authChain} \rangle_{\text{Usr}_1}, s'_{\text{Usr}_2} = \text{Sig}(\text{sk}(\text{Usr}_2), m'_{\text{Usr}_2})$ 3. Usr_1 sets $\langle \text{authChain} \rangle_{\text{Usr}_1} = \{\langle m_{\text{Usr}}, s_{\text{Shr}} \rangle \langle \text{authChain} \rangle_{\text{Usr}_1}\}$

Fig. 3. Protocol procedures: rights procurement, car access and rights sharing

unit or an infotainment device). The user ID_{Usr} requests to car ID_{Car} a specific functionality func^* . The message contains some random value to ensure freshness $\text{rand}_{\text{Usr}}^{128}$ and in case this is the first time the user connects to the car it also contains the authorization chain $\langle \text{authChain} \rangle_{\text{Usr}}$. The authorization chain consists in the message containing the rights of the user and the signature of an authorized party. For users that have freshly received rights from the sharing center, the authorization chain is just $m_{\text{Usr}}, s_{\text{Shr}} = \text{Sig}(\text{sk}(\text{Shr}), m_{\text{Usr}})$. Other users may have acquired rights from a regular user as discussed next and will present the authorization chain resulting from the next protocol component. The car checks that the authorization chain is correct and the time intervals specified in it match the current time. The car replies by sending a random value $\text{rand}_{\text{Car}}^{128}$, authenticates and links this value to the previous values by signing, i.e., $s'_{\text{Car}} = \text{Sig}(\text{sk}(\text{Car}), m'_{\text{Car}} || m'_{\text{Usr}})$. The user confirms his identity by signing the received challenge, i.e., $s'_{\text{Usr}} = \text{Sig}(\text{sk}(\text{Usr}), m'_{\text{Usr}} || m'_{\text{Car}})$. If the signature is correct the car executes the corresponding functionality and confirms this to the user by a new signature, i.e., $s''_{\text{Car}} = \text{Sig}(\text{sk}(\text{Car}), s'_{\text{Usr}})$.

The *rights delegation* stage occurs over an insecure channel, e.g., Bluetooth or WiFi, between two smartphones. First, user Usr_1 requests particular functionalities from Usr_2 , this is done in identical manner as when asking functionalities from the sharing center. Then user Usr_2 , if he agrees to share his rights, will reply with a message containing a signature over the rights as well as his authorization chain $\langle \text{authChain} \rangle_{\text{Usr}_1}$

```

step trans4(X, Y, SID, ACT, PkUsr, PkCar, NC, NU) :=
state_car(1, X, Y, ACT, PkUsr, PkCar, NC, NU).
iknows(crypt(inv(PkUsr), pair(X, pair(Y, pair(ACT, pair(NU, NC))))))
=>
iknows(crypt(inv(PkCar),
              crypt(inv(PkUsr), pair(X, pair(Y, pair(ACT, pair(NU, NC))))))).
state_car(2, X, Y, ACT, PkUsr, PkCar, NC, NU)

```

Fig. 4. Code snippet for the last transition of the car access protocol in the AVISPA [1] IF format

which proves that he indeed has access to the corresponding functionalities. Subsequently Usr_1 sets his authorization chain as $\langle authChain \rangle_{Usr_1} = \{ \langle m_{Usr}, s_{Shr} \rangle \parallel \langle authChain \rangle_{Usr_1} \}$.

Security analysis. Our protocol is designed for the general case of a Dolev-Yao [7] adversary that has full control of the communication channel. Since we build upon regular cryptographic blocks (which are considered to be secure) a formal verification of the protocol should be sufficient in assessing its security. For this, we use the AVISPA platform [1] and model the protocol in the IF language. As model-checker we choose CLAtse [19] which is one of the AVISPA [1] back-ends. For brevity, we modeled only the car access sub-protocol II. Figure 4 contains the code snippet in IF for the last transition of the car access sub-protocol. Signatures are modeled in AVISPA as encryption with the inverse of the public-key, i.e., $crypt(inv(PkUsr), message)$. Messages can be formed under the *pair* operator and the entire communication is mediated by the intruder knowledge by the persistent fact *iknows* (this responds to the Dolev-Yao model since the intruder is the channel). Verifying the protocol consists in defining one action for the honest user, e.g., *open car*, and another for the adversary, e.g., *start engine*, and determine whether the car will execute the intruder action. The model-checker reported the protocol to be safe. Verifying the entire protocol suite may be subject of an extended version of our work.

4 Experiments

In this section we clarify experiments on the platforms of our setup: Raspberry Pi with Optiga TPM, Android devices and in-vehicle controllers.

4.1 Deployment on Infineon TriCore

While medium to high-end car models will benefit from the performance of infotainment unit processors which is comparable to that provided by smartphones, this is not the case for low-end models. To cover the low-cost vehicle sector we employ the Infineon AURIX TC297, an embedded platform dedicated to specific automotive functionalities such as powertrain, chassis and body.

The TC297 is equipped with three 32 bit cores optimized for signal processing each of which can operate at a top frequency of 300MHz. A total of 729 KBytes of RAM

and 8 MBytes of Flash are available on chip. Members of the AURIX family of micro-controllers can be equipped with a hardware security module (HSM) which provides a secure key storage and execution environment along with HW implemented True Random Number Generator (TRNG) and 128-bit AES. Since the chip provides no HW support for implementing RSA we based our implementation on Miracl (Multiprecision Integer and Rational Arithmetic Cryptographic Library) <https://github.com/miracl/MIRACL>.

We tested the computational performance of the TC297 in executing basic steps of the proposed protocol on a single core. An RSA signature is performed in 26 ms, while the verification is executed in 462 ms. The implementation requires 74 and 73 ms for executing the sign and verify steps respectively using Shamir's ID-based signature with a 2048 bit key.

4.2 TPM Simulator and the Optiga TPM on Raspberry Pi

As a first step to test the functionalities of TPM 2.0 we have installed tpm2-simulator [10], tpm2-tss [4], tpm2-abrmd [6] and tpm2-tools [5] on a 32-bit Ubuntu Linux running on a virtual machine.

To obtain a crisper image on TPM 2.0 functionalities, we first managed to send commands to the tpm2-simulator using tpm2-tools for the following operations: i) generate random numbers up to 48 bytes (limited by max hash size), ii) create a primary key by selecting the key type, hash method and hierarchy under which the key-pair is created, iii) create a local object under the primary key consisting of public and sensitive part of a new key-pair, iv) import the created object as transient in the TPM, v) link an OpenSSL generated key-pair to a primary key in the TPM, vi) import an OpenSSL key-pair in the TPM, vii) encrypt local files using the imported object, viii) decrypt the local files using the imported object, ix) make the transient object persistent in the TPM, x) generate the hash digest of local files.

Once all these operations were tested, we ensured that all the TPM functionalities required by our experiments are available on the Raspberry Pi 3B+ connected to an Infineon OPTIGA TPM2.0 evaluation board. For the hardware experiments with the Raspberry Pi we have used Raspbian Stretch Lite April 2019 with Kernel version 4.14. In order to be able to identify the Optiga TPM on the Raspberry Pi we had to patch and build the Raspbian Kernel of the Raspberry Pi following the application note from [14] by adding the TPM support and also the Infineon TPM board in the device tree overlay. Afterwards we updated the kernel on the microSD card of the Raspberry Pi and managed to communicate with the TPM board after installing tpm2-tss[4], tpm2-abrmd[6] and tpm2-tools[5].

Considering the commands sent to the simulated TPM in Linux, we have benchmarked the duration of each public-key operation performed by the Raspberry Pi and the OPTIGA TPM (neglecting the transmission time to and from the TPM). The measurement results are shown in Table 2.

Table 2. Operation time for TPM commands

Command	Output	Output size [bytes]	Duration [ms]
Create a RSA-2048 primary key	primary.ctx	1036	20736
Create an RSA-2048 encryption key	key.pub, key.priv	280, 192	237
Load an RSA-2048 encryption key	object.ctx	1032	227
Perform RSA encryption	file.encrypted	256	164
Perform RSA decryption	file.decrypted	256	326

Table 3. Execution time for operations of the Shamir signature on the evaluated platforms

Operation	Platform			
	TPM	Raspberry Pi 3 B+	Samsung Note 8	Infineon TC297
Shamir IBS Gen	342 ms	n/a	n/a	n/a
Shamir IBS Sign	n/a	284 ms	5.3 ms	74ms
Shamir IBS Ver	n/a	50 ms	3.5 ms	73ms

4.3 Android Implementation

The Android implementation uses secret keys for the Shamir identity-based signature provided by the Optiga TPM module. The signing and verification functionalities are implemented using the Java BigInteger class. Currently the implementation is software based. An improvement on this (in terms of security) is to use the TPM as cryptographic co-processor. That is, while support for identity-based schemes does not exist on TPM, we can still perform modular exponentiations as regular RSA encryptions. However, the TPM implementation that we had does not allow loading large public exponents (the default is 65537). Thus, only the computations of $r^e \bmod n$ and $s^e \bmod n$ could be performed which are fast anyway (since e is small). We did not succeed in loading a arbitrary exponent h (which is the hash of the message) to compute $r^h \bmod n$ and $t^h \bmod n$ so the client/car-side implementation was entirely software based.

Tables 3 and 4 give an overview of the computational results on each of the platforms. Shamir’s identity-based signature is contrasted with regular RSA signatures. The computational results are graphically summarized in Figure 5.

5 Conclusion

A full scale implementation of our protocol may be subject to future work. The aim of this shorter communication was to establish whether the associated building blocks, e.g., identity-based crypto, and technologies, e.g., TPM, are within reach for an automotive scenario. Clearly, high-end in-vehicle controllers such as the Infineon TriCore are ready for public-key primitives and identity-based cryptography in particular. Mobile phones and single-board computers have even greater computational power and memory resources. As proved by our easy-to-use car sharing scenario, there are clear advantages in terms of flexibility when using these cryptographic primitives. The future

Table 4. Execution time for operations of the RSA signature on the evaluated platforms

Operation	Platform			
	TPM	Raspberry Pi 3 B+	Samsung Note 8	Infineon TC297
RSA Gen	220 ms (gen) + 220ms (load)	5.1 s	190 ms	n/a
RSA Sign	342 ms	121 ms	3.7 ms	462 ms
RSA Ver	198 ms	5.2 ms	0.3 ms	26 ms

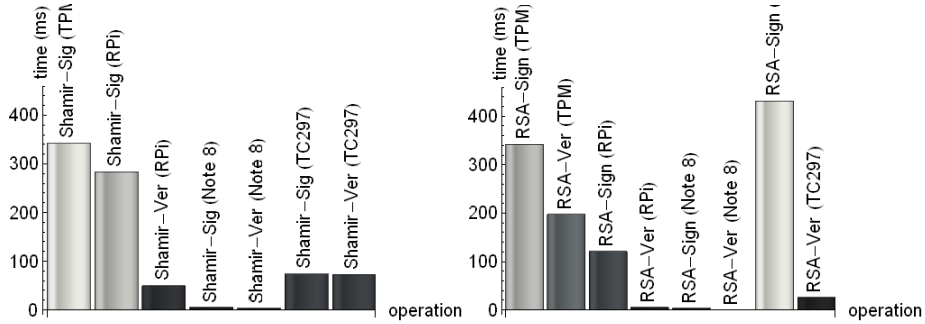


Fig. 5. Graphic summary of computational results for Shamir's signature (left) and regular RSA signature (right)

may bring single-board computers similar to Raspberry Pi inside cars if not delivered by the manufacturers then as a result of home projects. Aftermarket equipments are common in the automotive sector and DIY projects are also routine. Since CAN bus support exists for Raspberry Pi, turning this device into an in-vehicle body controller may not be a distant dream.

Acknowledgement. We thank the reviewers for helpful comments on our work. This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS-UEFISCDI, project number PN-III-P1-1.1.-TE-2016-1317 (2018-2020) <http://www.aut.upt.ro/~bgroza/projects/presence/>.

References

1. Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuéllar, J., Drielsma, P.H., Héam, P.C., Kouchnarenko, O., Mantovani, J., et al.: The avispa tool for the automated validation of internet security protocols and applications. In: International conference on computer aided verification. pp. 281–285. Springer (2005)
2. Brandl, H.: Trusted Computing: The TCG Trusted Platform Module Specification. In: Embedded Systems. New Munich Trade Fair Centre (2004), <http://opensug.net/utilisec/embedded/Shared%20Documents/Device%20Security/>

VariousInputs/ShrinathInputs/Basic_Knowledge_EC2004.pdf, Accessed: 2019-05-03

3. Busold, C., Taha, A., Wachsmann, C., Dmitrienko, A., Seudié, H., Sobhani, M., Sadeghi, A.R.: 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. pp. 233–242. ACM (2013)
4. community, D.: TCG TPM2 Software Stack. <https://github.com/tpm2-software/tpm2-tss>, accessed: 2019-04-15
5. community, D.: The source repository for the TPM (Trusted Platform Module) 2 tools. <https://github.com/tpm2-software/tpm2-tools>, accessed: 2019-04-15
6. community, D.: TPM2 Access Broker and Resource Management Daemon. <https://github.com/tpm2-software/tpm2-abrmd>, accessed: 2019-04-15
7. Dolev, D., Yao, A.: On the security of public key protocols. *IEEE Transactions on information theory* **29**(2), 198–208 (1983)
8. Francillon, A., Danev, B., Capkun, S.: Relay attacks on passive keyless entry and start systems in modern cars. In: NDSS (2011)
9. Glas, B., Sander, O., Stuckert, V., Muller-Glaser, K.D., Becker, J.: Car-to-car communication security on reconfigurable hardware. In: VTC Spring 2009-IEEE 69th Vehicular Technology Conference. pp. 1–5. IEEE (2009)
10. Goldman, K.: IBM’s Software TPM 2.0. <https://sourceforge.net/projects/ibmswtpm2/>, accessed: 2019-04-15
11. Guette, G., Bryce, C.: Using tpms to secure vehicular ad-hoc networks (vanets). In: IFIP International Workshop on Information Security Theory and Practices. pp. 106–116. Springer (2008)
12. Guillou, L.C., Quisquater, J.J.: A “paradoxical” identity-based signature scheme resulting from zero-knowledge. In: Proceedings on Advances in cryptology. pp. 216–231. Springer-Verlag (1990)
13. Guillou, L.C., Ugon, M., Quisquater, J.J.: Cryptographic authentication protocols for smart cards. *Computer Networks* **36**(4), 437–451 (2001)
14. Infineon: Optiga TPM SLB 9670XQ2.0. <https://www.infineon.com/cms/en/product/security-smart-card-solutions/optiga-embedded-security-solutions/optiga-tpm/slb-9670xq2.0/>, accessed: 2019-04-22
15. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Workshop on the theory and application of cryptographic techniques. pp. 47–53. Springer (1984)
16. Steger, M., Boano, C.A., Niedermayr, T., Karner, M., Hillebrand, J., Roemer, K., Rom, W.: An efficient and secure automotive wireless software update framework. *IEEE Transactions on Industrial Informatics* **14**(5), 2181–2193 (2018)
17. Symeonidis, I., Aly, A., Mustafa, M.A., Mennink, B., Dhooghe, S., Preneel, B.: Sepcar: A secure and privacy-enhancing protocol for car access provision. In: European Symposium on Research in Computer Security. pp. 475–493. Springer (2017)
18. Tillich, S., Wójcik, M.: Security analysis of an open car immobilizer protocol stack. In: Trusted Systems, pp. 83–94. Springer (2012)
19. Turuani, M.: The cl-atse protocol analyser. In: Intl. Conf. on Rewriting Techniques and Applications. pp. 277–286. Springer (2006)
20. Verdult, R., Garcia, F.D., Balasch, J.: Gone in 360 seconds: Hijacking with hitag2. In: Proceedings of the 21st USENIX conference on Security symposium. pp. 37–37. USENIX Association (2012)
21. Wei, Z., Yanjiang, Y., Wu, Y., Weng, J., Deng, R.H.: Hibs-ksharing: Hierarchical identity-based signature key sharing for automotive. *IEEE Access* **5**, 16314–16323 (2017)

22. Wetzels, J.: Broken keys to the kingdom: Security and privacy aspects of rfid-based car keys. arXiv preprint arXiv:1405.7424 (2014)
23. Wolf, M., Gendrullis, T.: Design, implementation, and evaluation of a vehicular hardware security module. In: International Conference on Information Security and Cryptology. pp. 302–318. Springer (2011)
24. Wolf, M., Weimerskirch, A., Paar, C.: Automotive digital rights management systems. In: Embedded Security in Cars, pp. 221–232. Springer (2006)