A Key to Embedded System Security: Locking and Unlocking Secrets with a Trusted Platform Module

Teri Lenard, Anastasija Collen and Niels A. Nijdam
Centre Universitaire d'Informatique,
Geneva School of Economics and Management,
University of Geneva, Switzerland
teri.lenard@unige.ch, anastasija.collen@unige.ch,
niels.nijdam@unige.ch

Bela Genge
Department of Electrical Engineering
and Information Technology, George Emil Palade
University of Medicine, Pharmacy, Science,
and Technology of Targu Mures, Romania
bela.genge@umfst.ro

***Abstract*—Security hardware modules were designed to provide a viable solution that can empower Embedded Systems (ES) with state-of-the-art cryptographic and security capabilities. They can execute cryptographic operations, securely store sensitive information, or provide measurements for attestation. A key element in designing and implementing security solutions on top of a security hardware, such as the Trusted Platform Module (TPM), is secure secret storage. The work at hand addresses the problem of secret protection by showcasing how the TPM standard can serve as a vault in protecting sensitive information in ES. This is accomplished as follows. Secrets are locked in the TPM according to Platform Configuration Register (PCR) policies created on top of the system state and sealing. In contrast, unlocking is achieved through TPM unsealing. In both cases, secure and authenticated sessions are enforced while communicating with the TPM. Furthermore, our work goes a step further and presents a simple TPM attestation protocol, destined to verify the system state and TPM application. Lastly, a series of experiments were conducted on a reference hardware, with two different TPM configurations, to measure execution times of TPM operations.**

***Index Terms*—trusted platform module, secure storage, embedded systems**

1. Introduction

Embedded Systems (ES) have built in the past decades the necessary foundation for designing and constructing robust and reliable systems that require a high level of automation, data processing, monitoring and control, and tolerance to faults and errors. Such systems range from Industrial Control Systems (ICS), Industrial Internet of Things (IIoT) to Automotive Systems (AS), to mention a few. A common characteristic across these systems, is that they were not built with security in mind [1]. This aspect was initially neglected since these systems were self-contained, isolated and disconnected from the outside world, having a narrower scope in terms of functionalities. This factor gave a certain level of security or assurance that the system is safe. However, once they became interconnected with the outside world (e.g., Internet) security became a significant issue. Consequently, this interconnectivity revealed a grim picture that was right away seen by malevolent parties.

As a counter measure, specially tailored security solutions and protocols were developed by the scientific community in tandem with the industry. This had as an outcome a plethora of anomaly and outlier detection techniques [2], data authentication [3] and key distribution protocols [4, 5], and intrusion detection and firewall systems [6]. Along this, secure hardware technologies such as Physical Unclonable Functions (PUFs) [7], Hardware Security Modules (HSMs) [8] and Trusted Platform Modules (TPMs) [9] became an attractive solution to strengthen the existing system security level of ES. Therefore, protecting and shielding application secrets, cryptographic keys, or certificate became a core building block in designing and implementing security protocols and applications.

The benefit brought up by a security enclave and cryptographic co-processor such as the TPM is significant. The TPM is a standard designed to enable a system with a powerful tool that can generate and store cryptographic keys, executes cryptographic operations isolated from the main processing unit, and enable equipment identification or attestation. While the TPM offers concrete guidelines on how Original Equipment Manufacturers (OEMs) should design the chip, in the hands of a user (e.g., security engineer, developer) it represents a tool that can be used to correctly secure a given system.

The work at hand demonstrates how an ES (e.g., automotive vehicle control unit or Internet of Things (IoT) sensor) that posses the capabilities of a TPM, can securely lock and unlock sensitive information stored in the TPM. Additionally, the proposed work goes a step further and exemplifies how the software managing the access control to this information can be remotely attested by a trusted authority with the TPM built-in features. Our work follows best practices in terms of TPM usage extracted from the standard [9], to enforce a policy-based access control to sensitive information, with properly encrypted and authenticated communication between the embedded device software and the TPM.

The remainder of the paper is structured as follows. In Section 2 we offer a set of background definitions and concepts on the TPM. Afterwards, Section 3 outlines the assumptions considered in our work. Following this, in Section 4 outlines the design of our solution. Subsequently, a set of performance measurements are given in Section 5, with the related work in Section 6. Lastly, our paper concludes Section 7.

2. Trusted Platform Module

The TPM [9] is a standard developed by Trusted Computing Group (TCG) [10], that encapsulates a tamper-proof environment dedicated to execute cryptographic operations in isolation from the main processing unit. TPMs serve as a trust anchor rooted in hardware, on top of which trusted security services can be built. TPMs allow secure key generation (e.g., via key hierarchies [11]) and storage (e.g., via non-volatile memory and sealing [12]), basic cryptographic operations (e.g., encryption, hashing, digital signature), trusted and measured boot [13], and remote attestation by leveraging its Platform Configuration Registers (PCRs) [14]. The presence of TPMs in the ES was deeply researched in the past decade. For instance, in automotive systems, OEMs such as Infineon is producing now TPMs dedicated to AS and IIoT. Additionally, TPM use-cases are offered not only by TCG [15], but by OEMs themselves [16, 17].

TPMs store and have the ability to generate several types of special cryptographic keys. The Storage Root Key (SRK) is one such special key. Rooted in the TPM hardware, the SRK is only available inside the TPM and it can never leave the TPM environment. The SRK is generated from a random seed, when a TPM user (e.g., OEM) takes ownership over the TPM. SRK has the purpose to protect system's application secrets.

Another key shielded by the TPM is the Endorsement Key (EK), which functions as a unique identity for the TPM, exposing the public part to the user through the TPM, while keeping the private part protected. The EK is generated by the OEM that produces the TPM (e.g., Infineon) using a secret seed within the TPM. With the endorsement hierarchy, TPMs allow the generation of persistent or ephemeral Attestation Identity Key (AIK), leveraged to sign and attest platform depended measurements.

Key hierarchies represent one method through which TPMs can generate and store cryptographic keys. Since TPMs have a limited amount of Non-volatile Memory (NVRAM), key hierarchies allow deterministic key generation starting from a master secret. The TPM has three persistent hierarchies. First, there is the platform hierarchy which is controlled by the platform manufacturer. Second, the storage hierarchy is managed by the platform owner. And thirdly, the endorsement hierarchy, which is a privacy sensitive tree intended for TPM attestation and identification. Additionally, TPMs have a more generic hierarchy, named the NULL hierarchy, which is dedicated to ephemeral keys and operations. In terms of data protection, TPMs have a mechanism called *sealing*. With sealing, the TPM can encrypt the private part of a key generated from a hierarchy and store it on the disk, while the public part remains available. This further links the private part of the key to the TPM, making it usable only with the TPM that sealed the key.

PCRs are a TPM-specific feature that provide hash-based measurements for remote attestation and trusted/measured boot. In a TPM, PCRs are grouped in different PCR banks according to their hash function (e.g., SHA-1, SHA-256) [14]. A PCR has a write-only state that stores a hash digest. PCRs support a single operation called *extend*. On system boot, the state of a PCR is set to a default value (e.g., a sequence of 0 or 1 bytes equal to the hash digest length). If an extend operation is performed on a PCR, the current state is concatenated with a new digest, and the new digest output is stored in the PCR state. By leveraging a bank of PCRs and a sequence of extend operations, hash-based measurements can provide a system state useful to determine if the system executes trusted code. Additionally, this system state can be leveraged to attest the system, with a quote operation. Quoting the PCR state implies reading a set of PCR values from the bank, and signing them with the AIK. The AIK is an anonymous key derived from the endorsement hierarchy to be used for privacy sensitive tasks, such as quoting PCRs, signing or certifying.

3. Assumptions

Protecting sensitive information with a TPM implies the presence of a TPM-based measured boot [18]. The measured boot process aims to verify the integrity of the system firmware, boot loader, kernel, and drivers, before each component is loaded in the memory. This verification ensures that only trusted code is executed on system start-up and was not modified by a malicious attack [19] (e.g., rootkit, bootkit). Measured boot is achieved using TPM PCR banks. A set of PCRs are extended before boot with the hash values of each system component that requires integrity checks (e.g., kernel). This PCR system state can confirm if the booted system is in an expected state (i.e., runs trusted code), and, depending on it, PCR access policies can be constructed. Additionally, PCR-based measured boot enables a trusted authority to remotely attest at run-time the system state.

Measured boot differs from other booting options such as secure or trusted boot. While trusted boot achieves similar properties with measured boot, without requiring the presence of a TPM, secure boot verifies the running code with digital signatures, and certificates emitted by a trusted authority. In the current work, measured boot is an essential feature, since the access control policy employed to protect sensitive information leverage the trusted system state. This approach assumes that the system itself must decide when it is in an expected secure state that allows secure access to the sensitive information stored in the TPM.

A limitation here is the fact that TPM measured boot assures system integrity at boot. During run-time, if an attacker is able to inject or remote execute code, this defence mechanism is not enough. This is also known for TPM remote attestation, where the PCR system state needs to attest the integrity of the running code.

Lastly, since ES encompass a wide range of devices, it is to mention that our work targets ESs operated by a microkernel or monolithic kernel based operating system, on top of which an application (i.e., user mode) exists.

4. Proposed work

The symbols and notations from Table 1 are used to describe the proposed work.

For setup, the TPM was bootstrapped with the necessary keys and certificate. Figure 1 outlines these steps.

TABLE 1: Table of symbols.

A	Attestor
V	Verifier
C	Certificate authority
pk	Public key
sk	Private key
ek	Endorsement key
ak	Attestation key
srk	Storage root key
Crt	Certificate
$\{\}$	Encryption or sealing
h	TPM persistent handle
c	TPM context
P	TPM set of PCRs values

Here, and in the rest of the paper, TPM command notation [20] is used to outline TPM operations. The set() operation denotes an initialisation procedure that sets a constant value to a TPM handle. In the TPM world, a handle points to an existing memory space in the TPM's NVRAM that can store a persistent object. Accordingly, three persistent TPM handles were defined: (1) h_{ek} corresponding to EK, (2) h_{srk} for SRK and (3) h_a corresponding to the AIK.

The EK and AIK were created under the endorsement hierarchy, and the SRK on the owner hierarchy. Since the EK and SRK are persistent keys generated from a secret seed inserted in the TPM by the TPM's OEM, the tpm2_createprimary(h) call was used for both cases. Here, h denotes a handle associated with a TPM hierarchy. In other words, this call signifies the creation of a TPM primary persistent key from a primary seed, on a specific hierarchy. The (4) tpm2_createprimary(h_{ek}) generates the EK on h_{ek} and returns its public key pk_{ek}. Likewise, (5) tpm2_createprimary(h_{srk}) call creates the SRK and returns the corresponding public key pk_{srk}. Afterwards, (6) tpm2_create(h_a) generates the AIK and returns the public part pk_a, the sealed private part $\{sk_a\}$ with the associated TPM context c_a. Compared to persistent handles, transient objects are only temporary loaded in the TPM, can be swapped in or out, and are identified by a context object. Lastly, (7) tpm2_getekcertificate() is called to retrieve from a trusted server the EK certificate. It is important to note that the key creation process might be done in parallel with the measured boot setup as part of system setup. Additionally, we showcase the TPM key creation process since not all TPM manufacturers ship their TPM with a generated EK, SRK or platform certificate.

Once the TPM is bootstrapped with the necessary keys, the access control policy can be set. Following the steps outlined in Figure 2, first, the set of PCR indexes associated with the measured boot were identified. Accordingly, let P denote the set of PCR indexes corresponding to this process. With (1) tpm2_pcrread(P) the state of the PCRs is read for policy creation. Using the TPM call (2) tpm2_policypcr(P), a TPM policy is created according to P. As this call will load a policy object into the TPM, as with other calls, a context c_P is returned. Afterwards, policy P can be linked with a newly created

```
TPM key creation.
1 : $h_{ek} \leftarrow\!\$\ $ set()
2 : $h_{srk} \leftarrow\!\$\ $ set()
3 : $h_a \leftarrow\!\$\ $ set()
4 : $pk_{ek} \leftarrow\!\$\ $ tpm2_createprimary($h_{ek}$)
5 : $pk_{srk} \leftarrow\!\$\ $ tpm2_createprimary($h_{srk}$)
6 : $c_a, pk_a, \{sk_a\} \leftarrow\!\$\ $ tpm2_create($h_a$)
7 : $Crt \leftarrow\!\$\ $ tpm2_getekcertificate()
```

Figure 1: Required cryptographic key creation procedure to setup the TPM.

TPM object through c_P. Before creating the sealed object, an authenticated and encrypted session is established with the SRK by calling (3) tpm2_startauthsession(). Following this, with (4) tpm2_create(), the policy context c_P, the SRK handle h_{srk}, and a secret s, s can be sealed with the TPM and only retrieved (e.g., unsealed) if policy P is met. This operation returns a pair of asymmetric keys pk_s and $\{sk_s\}$ derived from SRK, with sk_s sealed by SRK. Lastly, the transient policy object c_P is flushed from TPM's memory with the call (5) tpm2_flushcontext().

It can be observed that tpm2_create() was considered so far twice, once for AIK creation, and once to create a sealed object. According to [20], this specific call is meant to create a TPM object that can be loaded in the TPM (e.g., a pair of keys). Additionally, it can receive as parameter a sensitive information that can be sealed as well.

```
TPM sealing.
1 : $P \leftarrow\!\$\ $ tpm2_pcrread()
2 : $c_P \leftarrow\!\$\ $ tpm2_policypcr($P$)
3 : tpm2_startauthsession($c_P$, $h_{srk}$)
4 : $pk_s, \{sk_s, s\} \leftarrow\!\$\ $ tpm2_create($s, c_P, h_{srk}$)
5 : tpm2_flushcontext($c_P$)
```

Figure 2: Sealing a secret with the TPM and a PCR-based policy.

At run-time, when an application needs to retrieve and use the sealed secret, the procedure in Figure 3 is followed. Similar as in the sealing process, a policy object is first created in (1) and (2). Subsequently, the authentication session is established in (3), and with (4) tpm2_load(), pk_s, $\{sk_s, s\}$ is loaded into the TPM to obtain c_s. To unlock the sealed secret s, the (5) tpm2_unseal() call is used in conjunction with h_P. This call will succeed, only if policy P is met. Finally, in (6) and (7), tpm2_flushcontext() is called twice to remove the transient object c_s and c_P.

Attesting the integrity of an application interacting with a TPM represents the next step in the security chain. This is achieved via remote attestation [21]. This process implies three entities: an attestor A, a verifier V and a certificate authority C. The protocol starts with V sending a challenge n to A over a secure communication channel. Here, n denotes a random nonce. Afterwards, A proceeds to execute two TPM calls. The first one is a tpm2_quote(), which reads a designated PCR bank P (i.e., associated with measured boot and the code executed by

```
TPM unsealing.
1 :  $P \leftarrow\!\$$ tpm2_pcrread()
2 :  $c_P \leftarrow\!\$$ tpm2_policypcr($P$)
3 :  tpm2_startauthsession($h_P, h_{srk}$)
4 :  $c_s \leftarrow\!\$$ tpm2_load($pk_s, \{sk_s, s\}, h_{srk}$)
5 :  $s \leftarrow\!\$$ tpm2_unseal($h_P, c_s$)
6 :  tpm2_flushcontext($c_s$)
7 :  tpm2_flushcontext($c_P$)
```

Figure 3: Unsealing a secret with the TPM and a PCR-based policy.

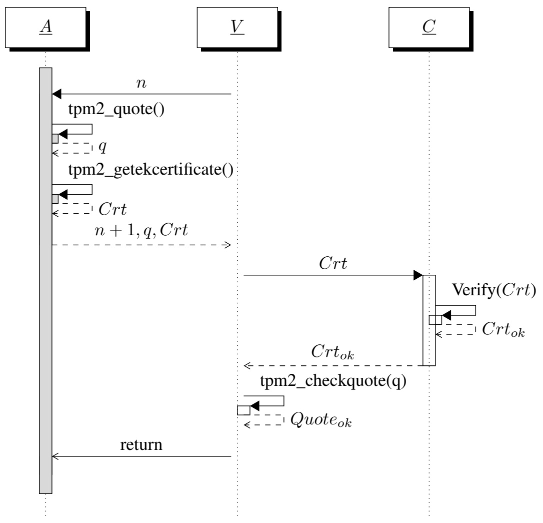

Figure 4: TPM-based attestation protocol.

A) and signs it with the AIK ak. Let this structure be denoted by q. Along quote q, A reads the EK Crt using tpm2_getekcertificate(). The EK certificate is unique to the TPM and is provisioned by the TPM manufacturer or OEM during system setup. A response back to V's challenge with an incremented nonce $n + 1$, quote q and Crt. Upon receiving this response, V proceeds to verify the authenticity of the certificate provided by asking C if Crt is valid. If C successfully verifies Crt, V then continues to verify q. To accomplish this, V is required to store locally in its TPM a representation of the expected PCR banks provided by A with q, and the public part of A's AIK. To complete the protocol, V executes a tpm2_checkquote() operation over q with the public key of A's AIK. If the verification is successful, the protocol finished as expected. All these steps can be visualised in the state diagram from Figure 4.

The presented approach achieves the simple goal of securely storing secrets, while guaranteeing additional security properties in an embedded ecosystem. First, the communication between the embedded application and the TPM is encrypted (i.e., symmetric encryption with SRK) and authenticated (i.e., via salted hashed message authenticated codes) with authenticated sessions. Secondly, by linking the sealed secret availability with a system state PCR policy, it is guaranteed that the system runs trusted code, and it is in an expected state when the secret is retrieved. Thirdly, since the secret is sealed in the TPM, reading the system image, does not leak any sensitive

Figure 5: Infineon OPTIGA SLB 9670 TPM 2.0 daughterboard for Raspberry Pi.

information. Lastly, to enable remote trusted services to attest the system state and the application accessing sensitive secrets, a simple attestation protocol was presented.

5. Performance measurements

The current section offers a set of performance measurements on TPM operations executed at run-time on the ES. These TPM operations are *unseal*, used to retrieve a TPM protected object, and *quote*, leveraged in attesting the system state. We did not consider measuring other TPM operation since this was already done in our lab in a previous work on the same hardware [22].

Two development boards were considered in our measurements, a Raspberry Pi (RPi) model 4 and a model 3B. Each RPi had attached an Infineon OPTIGA SLB 9670 TPM 2.0 daughterboard, as depicted in Figure 5. In terms of implementation, we considered the python module tpm2-pytss [1], which is a wrapper over the C tpm2-tss [2] library. For convenience and rapid prototyping, the high level Feature API (FAPI) was considered. FAPI offers a high level Application Programming Interface (API), abstracting low-level particularities through configuration parameters.

FAPI uses cryptographic profiles to configure the TPM with consistent usage of cryptographic algorithms, cryptographic key properties and templates, hashing algorithms, and PCR banks. By default, FAPI comes with two profiles, for ECC and for the RSA cryptosystem. We measured the execution times for the above mentioned TPM operations (e.g., quote and unseal), on both RPi boards with each profile. The intention here is to showcase on different hardware, and with different configurations, how the system behaves, outlining eventual delays introduced by the underlying operating system (e.g., raspberrypi os) on TPM function execution. For each experiment 100 measurements were recorded. Furthermore, in each case,

1. https://github.com/tpm2-software/tpm2-pytss
2. https://github.com/tpm2-software/tpm2-tss

TABLE 2: Minimum (Min.), maximum (Max.) and mean (Mean) execution times in seconds for quote and unseal TPM operations.

	RSA profile				ECC profile			
	RPi4		RPi3		RPi4		RPi3	
	unseal	quote	unseal	quote	unseal	quote	unseal	quote
Min.[s]	1.2018	0.5739	1.1983	0.5779	0.9629	0.3351	0.9650	0.3343
Max.[s]	1.2402	0.5912	1.2393	0.5926	1.0043	0.3466	1.0071	0.3462
Mean.[s]	1.2171	0.5847	1.2132	0.5861	0.9797	0.3391	0.9746	0.3390

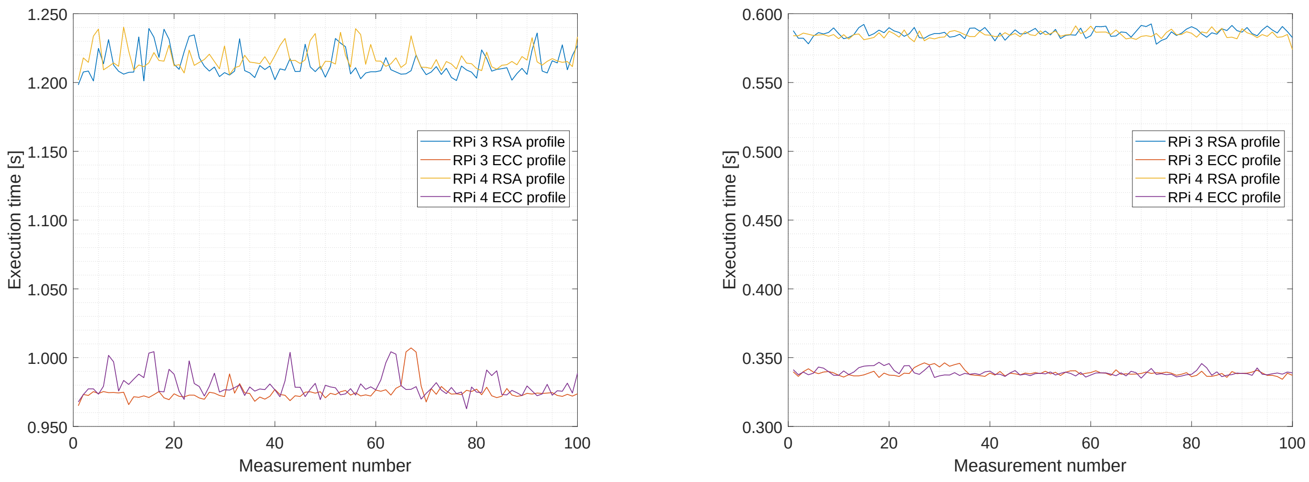

Figure 6: Execution times in seconds for TPM unseal (left) and quote (right) operation on the considered setup.

the minimum, maximum and mean execution time was computed.

As Figure 6 and Table 2 show, for both operations the underlying delay that is introduced by the operating system or the development board is not significant. For example, a quote operation on RPi4 with a RSA profile has a mean execution time of 0.5846s, while the same call on a RPi3 has a mean execution time of 0.5861s. Similarly can be stated for an unseal operation, that shows a mean time of 1.2171s in the case of a RSA profile on RPi4, and a mean time of 1.2132s in the case of RPi3. The only significant difference in execution times concerns the two FAPI profiles. On both RPi boards and TPM operations, the ECC FAPI profile showed faster execution times than the RSA one. For instance, on RPi4 the unseal operation with the ECC profile is 0.2384s faster than the RSA profile. Likewise, the statement holds for the mean execution time of the quote operation, which is 0.2456s faster on a ECC profile, than on a RSA one.

From these measurements it can be concluded that in terms of implementation, the underlying embedded system may influence to a certain degree, but not significantly, the execution of TPM operations. Furthermore, the system performance will be influenced in the end by the cryptosystem considered in the implementation. Lastly, it is to be mentioned that the measurements were conducted on a single TPM chip connected to the RPi via SPI. This may not be the case for real-world implementation, since the connection to the TPM may be done with a different communication channel.

6. Related studies

In the literature, TPMs were adopted as a solution to solve various security challenges. Integrating the TPM into an already existing system poses as consequence several challenges, as pointed out by Hoeller and Toegl [23]. The authors intended to determine the implications of integrating TPMs in cyber physical systems, and to point out relevant impacts on system safety and availability.

A secure access and feature activation system designed on top of TPMs for AS was proposed in [24]. The paper leverages the TPM as a trust anchor to enable secure authorisation policies. Prior work [22] likewise targets the domain of AS, where the TPM was considered to build security services (e.g., data authentication, key exchange) on top of it. On similar hardware as our prior work, Lu et al. [25] proposed an ES architecture composed of several protocol (e.g., synchronisation or access control protocols) designed on top of TPMs. In the same direction, Groza et al. [26] designed CarINA, an identity-based access control protocol that is built on top of TPM's features.

Other applications of TPMs can be found in the work of Gilles et al. [27] where the TPM was considered a proper solution to enable secure communication between IIoT devices, playing the role in parallel as medium for secure storage. Lu et al. [28] specifically targeted embedded IoT heath-care devices. Here, the authors extended the functionality of a TPM with a *shadow* TPM built in the form of a kernel module. Based on this scheme, three protocols were designed and implemented to preserve application integrity and authenticity.

Along the above mentioned TPM applications, the current work mainly fills the gap of locking and unlocking secrets with the TPM. The TPM acts as a secure medium through which ES can manage and restrict access to its secrets. Secret locking is achieved through PCR-based access policies derived from measure boot, and through TPM sealing calls. On the other hand, secret unlocking is done through unsealing. In both cases, authenticated communication sessions were enforced while communicating with the TPM.

7. Conclusion

Compared to the traditional information system that adopt TPMs as a root-of-trust for building security solutions, ES (e.g., AS, ICS, IIoT) raise different challenges in terms of TPM usage. In the ES ecosystems, the device itself is responsible to determine when the system is in a trusted state, allowing secure retrieval of sensitive information. Consequently, secure secret storage and policy-based access control represent a mandatory requirement. The work at hand addresses this aspect by showcasing how TPMs can be used to solve this problem. TPM features, such as sealing, policy PCRs and authenticated sessions, were employed. Additionally, a simple remote attestation protocol is presented to showcase how a remote trusted service can verify the integrity of the system state and application interacting with the TPM. Lastly, performance measurements were conducted on the most significant TPM operation (e.g., sealing, quote) executed at run-time on the embedded device, with different TPM configurations. The obtained results provide insights on the execution times for two distinct TPM modes of operation, namely with RSA and ECC cryptosystems.

Acknowledgements

This work is supported by the OPEVA project that has received funding within the Key Digital Technologies Joint Undertaking (KDT JU) from the European Union's Horizon Europe Programme under grant agreement No 101097267 and was co-funded by the Swiss State Secretariat for Education, Research and Innovation (SERI) and the Innosuisse – Swiss Innovation Agency. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or KDT JU. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] M. R. Asghar, Q. Hu, and S. Zeadally, "Cybersecurity in industrial control systems: Issues, technologies, and challenges," *Computer Networks*, vol. 165, p. 106946, 12 2019.

[2] D. Bhamare, M. Zolanvari, A. Erbad, R. Jain, K. Khan, and N. Meskin, "Cybersecurity for industrial control systems: A survey," *Computers & Security*, vol. 89, p. 101677, 2 2020.

[3] M. Agrawal, J. Zhou, and D. Chang, "A Survey on Lightweight Authenticated Encryption and Challenges for Securing Industrial IoT," 2019, pp. 71–94.

[4] B. Genge, P. Haller, A.-V. Duka, and H. Sándor, "A lightweight key generation scheme for end-to-end data authentication in Industrial Control Systems," *at - Automatisierungstechnik*, vol. 67, no. 5, pp. 417–428, 5 2019.

[5] Yee Wei Law, M. Palaniswami, G. Kounga, and A. Lo, "WAKE: Key management scheme for wide-area measurement systems in smart grid," *IEEE Communications Magazine*, vol. 51, no. 1, pp. 34–41, 1 2013.

[6] C. Young, J. Zambreno, H. Olufowobi, and G. Bloom, "Survey of Automotive Controller Area Network Intrusion Detection Systems," *IEEE Design & Test*, vol. 36, no. 6, pp. 48–55, 12 2019.

[7] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, "Physical Unclonable Functions and Applications: A Tutorial," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1126–1141, 8 2014.

[8] C. Labrado and H. Thapliyal, "Hardware Security Primitives for Vehicles," *IEEE Consumer Electronics Magazine*, vol. 8, no. 6, pp. 99–103, 11 2019.

[9] Trusted Computing Group, "Trusted Platform Module Library Specification, Family "2.0", Level 00, Revision 01.59," 2019.

[10] ——, "TCG TPM 2.0 Automotive Thin Profile For TPM Family 2.0; Level 0," 2018. [Online]. Available: https://trustedcomputinggroup.org/resource/tcg-tpm-2-0-library-profile-for-automotive-thin/

[11] Arthur Willand Challener and Davidand Goldman Kenneth, "Hierarchies," in *A Practical Guide to TPM 2.0: Using the New Trusted Platform Module in the New Age of Security*. Berkeley, CA: Apress, 2015, pp. 105–118. [Online]. Available: https://doi.org/10.1007/978-1-4302-6584-9_9

[12] ——, "NV Indexes," in *A Practical Guide to TPM 2.0: Using the New Trusted Platform Module in the New Age of Security*. Berkeley, CA: Apress, 2015, pp. 137–150. [Online]. Available: https://doi.org/10.1007/978-1-4302-6584-9_11

[13] S. Sanwald, L. Kaneti, M. Stöttinger, and M. Böhner, "Secure Boot Revisited: Challenges for Secure Implementations in the Automotive Domain," *SAE International Journal of Transportation Cybersecurity and Privacy*, vol. 2, no. 2, pp. 11–02, 8 2020.

[14] Arthur Willand Challener and Davidand Goldman Kenneth, "Platform Configuration Registers," in *A Practical Guide to TPM 2.0: Using the New Trusted Platform Module in the New Age of Security*. Berkeley, CA: Apress, 2015, pp. 151–161. [Online]. Available: https://doi.org/10.1007/978-1-4302-6584-9_12

[15] T. C. Group, "TCG TPM 2.0 Automotive Thin Profile For TPM Family 2.0; Level 0," 2020. [Online]. Available: https://trustedcomputinggroup.org/resource/tcg-tpm-2-0-library-profile-for-automotive-thin/

[16] Infineon Technologies AG, "Automotive application guide," 2019. [Online]. Available: https://www.infineon.com/dgdl/Infineon-Automotive-Application-Guide-2021--v02_00-EN.pdf?fileId=5546d462584d1d4a015891808e617573

[17] ——, "TPM 2.0 Protected IIoT Security Gateway for Industrial Control Systems ." [Online]. Available: https://www.infineon.com/dgdl/Infineon-ISPN_UseCase_Lanner_TPM_Protected_

Gateway-ApplicationBrochure-v02_00-EN.pdf?fileId=5546d4626c1f3dc3016c85fc875c00db

[18] O. Khalid, C. Rolfes, and A. Ibing, "On implementing trusted boot for embedded systems," in *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. IEEE, 6 2013, pp. 75–80.

[19] R. Wilkins and B. Richardson, "UEFI secure boot in modern computer security solutions," in *UEFI forum*, 2013, pp. 1–10.

[20] Trusted Computing Group, "Trusted Platform Module Library: Part 3 Commands, Family "2.0" Level 00 Revision 01.38," 9 2016. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-3-Commands-01.38.pdf

[21] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O'Hanlon, J. Ramsdell, A. Segall, J. Sheehy, and B. Sniffen, "Principles of remote attestation," *International Journal of Information Security*, vol. 10, no. 2, pp. 63–81, 6 2011.

[22] T. Lenard, B. Genge, P. Haller, A. Collen, and N. A. Nijdam, "An Automotive Reference Testbed with Trusted Security Services," *Electronics*, vol. 12, no. 4, p. 888, 2 2023.

[23] A. Hoeller and R. Toegl, "Trusted Platform Modules in Cyber-Physical Systems: On the Interference Between Security and Dependability," in *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 4 2018, pp. 136–144.

[24] C. Plappert, L. Jäger, and A. Fuchs, "Secure Role and Rights Management for Automotive Access and Feature Activation," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. New York, NY, USA: ACM, 5 2021, pp. 227–241.

[25] D. Lu, R. Han, Y. Wang, Y. Wang, X. Dong, X. Ma, T. Li, and J. Ma, "A secured TPM integration scheme towards smart embedded system based collaboration network," *Computers & Security*, vol. 97, p. 101922, 10 2020.

[26] B. Groza, L. Popa, and P.-S. Murvay, "CarINA - Car Sharing with IdeNtity Based Access Control Re-enforced by TPM," 2019, pp. 210–222.

[27] O. Gilles, D. Gracia Pérez, P.-A. Brameret, and V. Lacroix, "Securing IIoT communications using OPC UA PubSub and Trusted Platform Modules," *Journal of Systems Architecture*, vol. 134, p. 102797, 1 2023.

[28] D. Lu, R. Han, Y. Shen, X. Dong, J. Ma, X. Du, and M. Guizani, "xTSeH: A Trusted Platform Module Sharing Scheme Towards Smart IoT-eHealth Devices," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 2, pp. 370–383, 2 2021.