Concept of Smart Building Cyber-physical Systems Including Tamper Resistant Endpoints

1st Andreas Puesche
IDiAL
University of Applied Sciences and Arts Dortmund
Dortmund, Germany
andreas.puesche@fh-dortmund.de

2nd David Bothe
Institute for Internet Security
University of Applied Sciences Gelsenkirchen
Gelsenkirchen, Germany
bothe@internet-sicherheit.de

3rd Marco Niemeyer
Institute of communication technologies
University of Applied Sciences and Arts Dortmund
Dortmund, Germany
marco.niemeyer@fh-dortmund.de

4th Sabine Sachweh
IDiAL
University of Applied Sciences and Arts Dortmund
Dortmund, Germany
sabine.sachweh@fh-dortmund.de

5th Norbert Pohlmann
Institute for Internet Security
University of Applied Sciences Gelsenkirchen
Gelsenkirchen, Germany
pohlmann@internet-sicherheit.de

6th Ingo Kunold
Institute of communication technologies
University of Applied Sciences and Arts Dortmund
Dortmund, Germany
kunold@fh-dortmund.de

***Abstract*—Cyber-physical systems (CPS) and their *Internet of things (IoT)* components are repeatedly subject to various attacks targeting weaknesses in their firmware. For that reason emerges an imminent demand for secure update mechanisms that not only include specific systems but cover all parts of the critical infrastructure. In this paper we introduce a theoretical concept for a secure CPS device update and verification mechanism and provide information on handling hardware-based security incorporating trusted platform modules (TPM) on those CPS devices. We will describe secure communication channels by state of the art technology and also integrity measurement mechanisms to ensure the system is in a known state. In addition, a multi-level fail-over concept is presented, ensuring continuous patching to minimize the necessity of restarting those systems.**

***Index Terms*—Internet of Things, Cyber-physical Systems, Computer Science, IT-Security, Trusted Platform Module, OAuth2.0, TLS 1.3**

I. Introduction

In modern internet landscapes, innovations in the internet of things (IoT) and cyber-physical systems (CPS) sector keep on sprouting. According to [1], nearly 20 percent of organizations encountered at least one IoT-based attack in the recent three years and the forecast predicts a growth in expenses to implement endpoint security from $240 million in 2016 to $631 million in 2021. Observing this development in the IoT and CPS market, the near future will sculpt those IoT and CPS landscapes with device automation gaining more ground in technology driven living environments.

CPS are a combination of physical and computational components and present a reflection of a real world entity, incorporating sensors and actuators to process domain specific data while building a network of interacting elements. CPS that are connected to other CPS or any remote party contain IoT components [2] that communicate over untrusted networks.

A single CPS or compositions of CPS can form smart homes and buildings which are capable of automatizing common control engineering functions like heater, power and airflow management, control engineering and even surveillance. While smart homes are generally understood as the automation of private homes, smart buildings target multiple homes and also single-purpose buildings like corporate sites and government buildings, which incorporate even more functionality in their CPS. The main difference of smart homes and buildings is the concern of data privacy. While smart homes are a self induced optimization of private property, smart buildings face legal issues and are obliged to implement mechanisms for privacy and data protection. As our civilization becomes more dependant on such technology, a mandatory need arises to enhance the security on those devices and services [3]. In this regard, we will also consider the interference between security and dependability [4] while using hardware based security and apply it to our presented concept.

To mitigate risks due to firmware attacks on IoT components of CPS, we will introduce secure and smart update mechanisms covering the necessary parts of the systems infrastructure. The update mechanisms include the usage of a Trusted Platform Module 2.0 (TPM) [5] on each endpoint of the system to attestate the endpoints hard- and software configuration. Maintaining the integrity of each endpoint is essential to guarantee authenticity of the transfered data. Substituting invalid data for valid data can compromise the whole CPS. False data injection (FDI) primarely targets CPS and poses an unacceptable threat to each stage of such system. It is even possible to hide information in those systems by manipulating actuator states [6]. While controlling the smart building, a cloud system needs to collect data from each CPS to change states and behaviour of the CPS's actuators.

978-1-7281-1154-4/18/$31.00 ©2018 IEEE

In case of apartment buildings, each CPS belongs to one household and acts as a user with access to their specific data. This also applies to other forms of smart buildings where a central trusted entity needs to gain access to specific data for coordination, overview and maintenance of the whole building. To cover privacy and data protection, we will introduce a concept based on OAuth2.0 to authenticate each of the CPS endpoints and make the cloud data available to each respective user of the system.

II. Related Work

Available literature addresses security, authentication and update mechanisms including the integration of a TPM in IoT and CPS and their impact on dependability. The works provide background on our proposed concept to integrate hardware based security on large distributed systems and dealing with restrictions during update processes.

A. Impact of Security on Dependability

Smart buildings and other large CPS require to operate continuosly and guarantee some degree of dependability in regards of safety concerns. Security has always had an impact on dependability. Even on monolithic systems, the more security mechanisms are implemented, the more complex the operation of such system will get. Adding more operational complexity can introduce unwanted side effects on safety, e.g. immidiate shutdown processes are slowed down due to complex authentication mechanisms. [4] is concerned with the interference between security and dependability in CPS while employing a TPM and provides assessments on using different TPM functions in that environment. [7] discusses an *Agent-based Autonomic Cover (AAC)* approach to CPS to create a dependable system.

B. Authentication and Authorization in CPS

Regarding authentication and authorization in the CPS, standardized protocols and secure cryptographic methods are used. This ensures that the system is protected against vulnerabilities by appropriate measures. A successful attack on a system with insufficient authentication methods can result in data loss of sensitive private data as well as unauthorized access to physical actuators, potentially causing extensive damage. To avoid such frauds, it is recommended to follow the specifications of the Open Web Application Security Project for the implementation of authentication mechanisms. Here, the most common vulnerabilities [8] have been listed with corresponding counteractive measures.

[9] uses OAuth 2.0 to protect an IoT Network by implementing a security manager server that provides an authentication service for multiple IoT networks. [10] proposes an architecture forming an authorization framework that can be integrated by invoking an external *oauth-based authorization service (OAS)*.

C. Software Updates in CPS

CPS are part of a large distributed infrastructure. In this regard, there are some requirements to the overall architecture that need to be met. The states of the physical process need to be measured by collecting data through sensors. A CPS is then able to react to changes by modifying the state of their actuators, e.g. using more or less power to propel an electric motor in a real world entity. The CPS needs to exchange data frequently and reliable, requiring a highly scalable infrastructure. To maintain a specific Quality of Service (QoS), the CPS needs to also meet other requirements like asynchronous distribution and time constraints. Updates should only have a minimal impact on the reliability of CPS and the architecture must avoid becoming a lying endpoint as described in [11]. [12] presents a system architecture to update CPS components by *Dynamic Software Updating (DSU)*. [13] provides an alternative approach on DSU.

III. Basic Hardware Security Mechanisms

This section will provide brief information about the security mechanisms and measures that are incorporated into this concept. It will help to understand the whole update mechanism described in section IV where the integrity measurements as well as the update mechanisms are described in detail. Although Hardware Security Modules (HSM) are well suited for this purpose, the availability, robustness and low costs of a TPM made them the hardware of choice for CPS endpoints in this concept.

A. Trusted Platform Module

TPMs are passive security chips and incorporate mechanisms for trusted computing. The functionalities are often embedded in cryptographic context. This chapter will cover the most relevant use cases of a TPM and how they can be implemented in a CPS. As the concept opts for a dependable system, functions like *Secure Boot* are not described, due to the great loss in availability [4] to a distributed CPS. In general, a TPM can be used to implement cryptographic measures to protect a system or single software components against digital threads [14]. A TPM is also capable of generating cryptographic keys as well as storing them in a secure location. The TPM encapsules a tiny cryptographic co-processor for encryption and decryption and provides a *True Random Number Generator*.

B. Platform Configuration Registers

In case of the following update scenario, the TPM is used to measure the systems hard- and software configuration to prevent unwanted behavior of each endpoint, thus minimizing the risk of unwanted software updates that would overwrite the clients current configuration. This is accomplished by utilitzing the *Platform Configuration Registers (PCR)* of the TPM. The TPM records the current system state by cryptographically hashing software components and hardware information. The resulting value is stored in one of the PCRs. On each platform reboot, the PCRs are reset. The *TPM2_PCR_EXTEND*

operation is the only possibility to write into a PCR. As the name suggests, the value in a PCR can only be extended by another hash value. This operation is non-invertible and non-commutative.

C. Measured Boot

Measured Boot records the boot process of a platform without interrupting it. This is accomplished by extending the PCRs without further validating and storing the hashes of each measurement within the PCRs. In contrast to *Secured Boot*, the hashes are not validated each step of the boot chain before continueing and are simply extended. The resulting value can provide reliable information wether the boot process has been tampered or not.

D. Remote Attestation

In a distributed system, a remote party is usually not able to measure the software state of a platform. Reporting a platforms state purely by software implementation may result in a compromised *reporting software* that lies about its measurements. TPM attestation can generate cryptographic proof of a platforms state by utilizing PCRs. Extending the PCRs while successively measuring the software at device boot generates a *TPM quote* and signs it with a key from the TPM. The remote party can then verify that the signing key came from an authentic TPM and the report was unaltered.

E. TPM Keys

A common use of a TPM is to securely store keys. Keys can be stored hierarchically where a primary key acts as the root key of that hierarchy. This primary key never leaves the TPM unencrypted. Subsequent keys form a tree where each key is a child key from a parent key with the TPM's private *Storage Root Key* as the first parent. A parent key is an encryption key and encrypts child keys before they leave the secure environment of the TPM. This can be utilized to store keys in unsecure locations, e.g. on the device itself. When used, the keys need to be decrypted by their respective parent first which eventually leads to the primary key stored in the TPM itself. As an exception, the *Storage Root Key* is the only key never leaving the chip.

F. Transport Layer Security

The *Transport Layer Security (TLS)* protocol grants client/server applications a secure way to communicate over untrusted networks. With the release of TLS1.3 [15], support for legacy algorithms has been dropped and TLS defaults to state-of-the-art algorithms. The protocol enforces to use an *Elliptic Curve Diffie-Hellman (ECDH)* handshake to negotiate a shared secret (Key). A succesful handshake allows the peers to use this secret for protection of their application-layer traffic. Endpoints implementing a TPM can provide true randomness for the key generation procedure. This is done through the TPM's own *True Random Number Generator* without relying on external sources of randomness. Additionally, the TPM provides a secure storage for the generated keys.

IV. System Architecture

The technologies described above can be combined to form a new system architecture design and depict integrity aware update mechanisms. In this section, we will describe our concept on a secure architecture for CPS endpoints that allow for dynamic and secure update processes while minimizing system reboots on those endpoints and maximizing endpoint credibility.

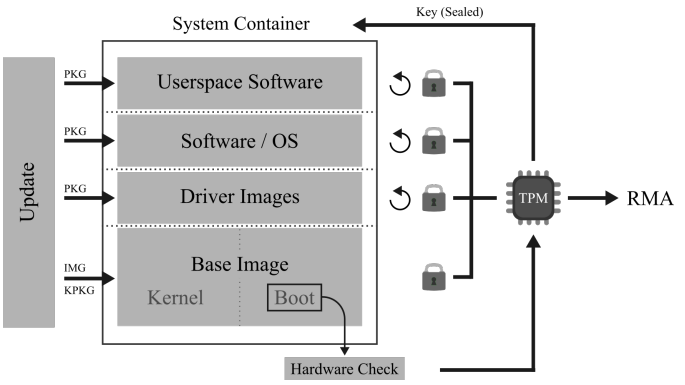

Fig. 1. Basic system architecture.

A. Basic Container Architecture

The system architecture consists of four separate containers that should be encrypted to prevent data leakage. Figure 1 illustrates the basic system architecture and its containers. The *Base Image* container provides the kernel and ramdisk of the operating system, *Driver Images* holds kernel drivers, *OS Software* contains the operating system components and the *Userspace Software* container hosts the controlled component managamenet software. Together they form the *System Container Stack*. To prevent system inoperability caused by defects during updates, each of the containers keeps their own *Mirror Container*, providing the last known functional software versions. Occuring startup errors will drive the systems to switch back to its preserved version of the *System Container Stack*.

After a successful update, the mirror images are re-written with the updated version. With the exception of the *Base Image*, the containers implement a package based mechanism (PKG) to dynamically push updates onto them. The *Base Image* will only be updated by providing a completely new image file (IMG) of that container and is exchanged as a whole. The kernel itself however may also be updated though *Kernel Update Packages* (KPKG) that are used to provide different kernel updates, which leads to updating the system without the need of rebooting the endpoint. Both, the regular and mirrored *System Container Stack* share an encrypted container on top which inherits configurations and meta information for all other components. This shared container is not illustrated here.

The *Base Image* itself must be encrypted as it represents the *Root of Measurement* for all components in the boot

chain and must not be manipulated under any circumstances. Any other container has to provide some integrity mechanism as described in IV-B2. This mechanism ensures a hardware supported bottom up check of the system through a hardware security module like a TPM as designated here. By reaching the end of the chain, the *Measured Boot* process is completed including all relevant components included in the measurement. Section III-D offers a description on how this can be used with *Remote Attestation* (RMA) to attestate the system's integrity to other parties.

B. Integrity Measurement Components

With hardware supported integrity checks of the systems state, the update mechanism gains more complexity due to profound changes in the platforms software configuration. To ensure a frictionless update of the components, the TPM features from section III must be taken into consideration and integrated as required.
The integrity measurement und corresponding update mechanism currently is a theroetical concept.

1) Integrity Aware Update Sequence: Keeping the systems integrity in a well known state, any carried out modification, e.g. through software updates, must be tracked to provide integrity measurement reference data. The system's integrity can then be checked against this reference data on a posterior point of time. We assume that all updates are provided through a secured channel, e.g. TLS connection, and contain the updated files as well as the respective cryptographic signature for each modification. This can be achieved by extending the PCR of a TPM. This allows us to discard the update server integrity as it only distributes packages and their integrity is completely depending on the cryptographic signature. If packages must remain private for some reasons, some login mechanism to the update server should be considered, which then can again be secured by an remote attestation of the server. This will not be covered by this paper.

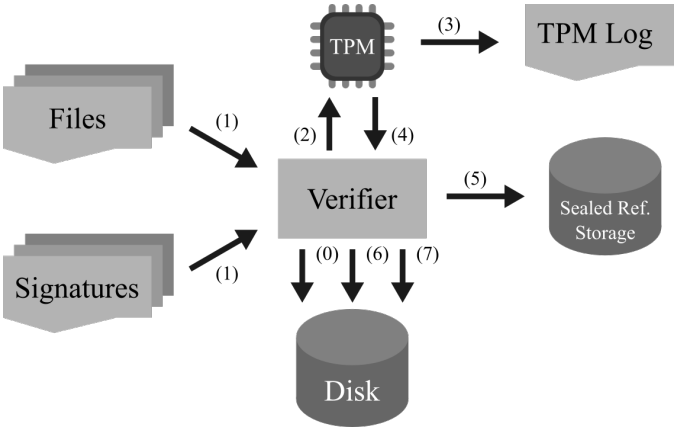

Fig. 2. Update procedure with integrity check and integrity reference database update.

The update procedure, illustrated in figure 2, operates as described in detail by each step**(n)**: As the main integrity sofware component, we introduce a **Verifier** that needs to be triggered on each *write()* call to the system kernel (system call). To provide a locking mechanism that prevents modifications of the target file **(0)** on any integrity enabled partition or device, the **Verifier** intercepts and wraps the default kernel functions. Each update must provide the file to be written, as well as its corresponding digital signature **(1)**. With the help of the **TPM**, the signature is checked for consistency by sending it along the file's checksum to the TPM. It will then be processed for verification V by calculating $V(pk_S, h_f, S(sk_S, h_f)) = accepted$ **(2)**, with pk_S as the software distributor's public key, h_f as the file's checksum HASH function, sk_S the distributor's secret key and S the files signature.

If the integrity has been successfully verified, the new integrity reference data needs to be persisted in an separately sealed storage secured through the TPM **(5)** by processing $[h_{path}, E(pk_{ts}, [s_{path}, h_{path}, S_f, D_m])]$. E being the encryption function requiring the TPM's used storage public key pk_{ts} for the concatinated path string s_{path}, h_{path} to conduct integrity checks, the file signature S_f provided by step **(4)** and further metadata D_m. The encryption will be done using the *TPM_seal* function. At this step, the TPM's PCR will be extended by calculating HASH H of PCR_n as $H_{pcr}(H_n|H_{n-1})$ which also includes an additional TPM Log entry **(3)**. The key advantage of this method is the ability to pre-calculate the post update system state, which then can be validated by remote parties through the TPM log and its PCRs as mentioned in section III. Once the reference data is persisted, the file needs to be written to the storage device at the desired location **(6)**. To avoid any concurrent access to the new file, the sequence must be atomic, ensuring that no ambiguous state occurs. A locking mechanism will cover this issue and must be released in the final step **(7)**.

The **Verifier** must be part of the kernel and restricted from userspace interference to harden the system. The update service running in the user mode has to be separated from any other application through *Mandatory Access Control (MAC)*, preventing it to be influenced. The system will only accept signed files from a trusted and known authority, so the provided files are authentic and no modification has occured. As older versions of the TPM are known to be vulnerable for certain attacks, it is highly recommendet to use chips of version 2.0 although there are recent known attacks as described in [16].

2) Access Triggered Integrity Verification: The file verification procedure has to be triggered at runtime on the *open()* system call to efficiently prevent the system from directly loading unverified content or applications. Figure 3 illustrates the necessary steps beginning with the *open* call **(1)** to the systems integrity enabled storage. This call will invoke the internal **Verifier** and assigns it to check the desired file. The **Verifier** then starts by reading the reference data from the sealed storage **(2)** and calling the TPM to decrypt the entry $[h_{path}, E(pk_{ts}, [s_{path}, h_{path}, S_f, D_m])]$ **(3)** to make $[s_{path}, h_{path}, S_f, D_m]$ accessible again **(4)**.

Step **(5)** takes the newly calculated checksum of the file

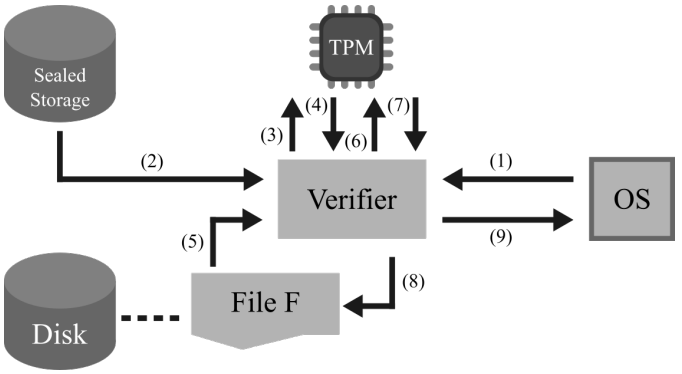

Fig. 3. File reading procedure with integrity verification.

and delivers it to the signature verification function V of the TPM. This is handled together with the reference signature to check the file's integrity **(6)** by $V(sk_S, h_f, S_f) = accepted$. On a positive verification result **(7)**, the **Verifier** allows the system kernel to deliver the file's content to the application requesting to open it **(8)(9)**. Just as processed in the update sequence in section IV-B1, these steps need to be atomic by locking the file to prevent intermediate modifications. This lock needs to be sustained until the kernel delivers the file's data **(9)**. Due to losses in the system's speed on consitantly accessed files, a caching mechanism may keep the contents at hold for future access, even if the file is closed temporary. This prevents recurrent checks to the same files. The time for caching should be chosen reasonable, due to longer caching times enlarging the timeframe where in-memory manipulation of the content is possible.

The credibility of the endpoint can then be proven by requesting an remote attestation. This attestation then will represent the latest state of the system after loading all files. Runtime manipulation cannot be uncovered through the access triggered integrity verifiation procedure as it is an in-memory only change. As depandability aspects prohibit a Secure Boot, the system may even load manipulated files on boot, which can only be detected by third parties through an attestation request of teh endpoint.

V. CPS Cloud

The widespread use of IoT systems necessitates the consistent implementation of architectural concepts that ensure functionality and system security. For this, methods such as "Security by Design" and "Privacy by Design" for Internet-based systems are required. With the previously described update mechanisms we create a basis for the following high-level security features. The explanation focuses on software security. Large CPS often include a central cloud component in their system architecture. Here, the cloud follows the approach of a service-oriented distributed architecture. It is common to provide an API which is implemented as a RESTful web service. The main task of the cloud is to identify, control and manage the connected sensors and actuators of each CPS. The implementation of data protection regulations combined with technical security measures increases the acceptance of the service-use of the end user. This should also be in the interest of the service provider because in most cases this is also the basis of their business models. Figure 4 depicts a brief network architecture overview, including the main components and the basic attestation sequence in between. The given setup uses a *Trusted Third Party* (TPP) with temporary leases to handle attestation requests and reports between the cloud components and the endpoints.

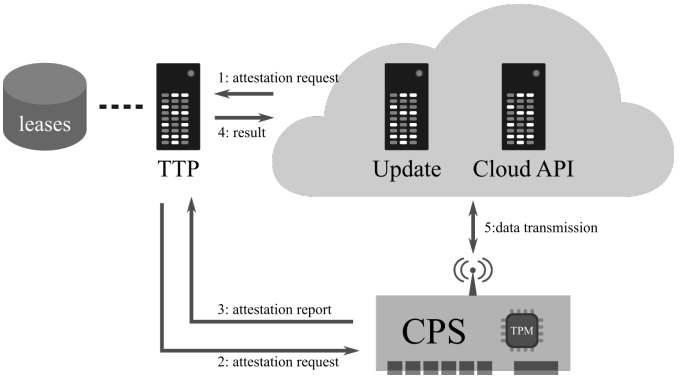

Fig. 4. Network architecture.

A. Software and Firmware Updates

Regarding software and firmware updates, the CPS-Cloud needs to meet some requirements. This includes the distribution of encrypted update files via a secure connection to the corresponding CPS. The update should be provided with a signature that can be verified by the CPS. This enables the maintenance of the distributed systems. Updates may be required to fix software errors, integrate new features, or fix known security vulnerabilities [17]. If a firmware update is required due to a security vulnerability, this update must be rolled out to all distributed systems in the shortest possible time. Data integrity checking is required to completely eliminate unwanted modification of a firmware update. If an integrity check is not executed, a software update may be subsequently provided with malicious code. In this case a modified configuration or software update can compromise the entire system. This concept of distribution has already been prototypically implemented by using an integrated broker system with secured transmission channels.

B. Digital Signatures

Digital signatures are commonly used to ensure that a software update is unchanged. Different software signature algorithms can be used to sign software components. These differ in terms of security level of the crypto system, hash algorithm, purpose and performance. Each year, the German Federal Network Agency provides a list [18] of cryptographic algorithms for the generation of qualified electronic signatures.

C. Authorization

Authorization of access to the API should be implemented via OAuth 2.0. For example a resource owner can define a scope for a dataset that should be made accessible for third parties by prior authorization. A resource owner is registered at the Identity and Access Management (IAM) of the CPS-cloud and he automatically obtains administrative privileges to manage his resources (e.g. sensors and actuators). Furthermore he can create new users and assign specific permissions to resources for every user. The privacy policy of the CPS-cloud ensures that the collected data will only be transmitted to other users or services with the prior consent of the owner.

D. Data Storage

The system architecture provides separate databases to store different types of data. On the one hand, measurement and status data are stored. It should be noted that the incoming data must be cyclically checked for integrity before it is stored. On the other hand, credentials are stored in a separate database with high security requirements. Passwords should never be stored unencrypted so passwords are stored as salted hash values. This is done using state of the art hash algorithms like SHA2-512 or SHA3-512. In addition, there is a database for device configuration and firmware updates. The assignment of the measurement and status data to the respective user is realized by pseudonymization.

E. Cloud API

The access to a resource is done over a two-level authorization. If a request is sent to a resource of a specific API the user context within the CPS cloud is resolved via an access token (e.g. JSON web token). This completes the first step of the Access Control. In the next step, the permission is checked for the corresponding resource. For example, if a switching command is send to an actuator, a write permission must exist for this resource. For this purpose an API gateway is used to realize this presented concept.

VI. Conclusion

Providing security mechanisms in large distributed systems is a mandatory task when critical data is exchanged. In case of smart buildings, there are also many safety concerns, especially when real life entities like huge machinery in CPS are involved. In addition, privacy concerns arise and need to be processed without data leakage. In our concept we covered the basic security mechanisms to react to such requirements. This paper presents a solution to overcome this obstacles by providing ideas that combine well tested mechanisms in a different way. Dependability will always counteract security mechanisms in one way or another but can build a strong combination when kept in balance. In CPS environments, dependability and security both have a great impact on the requirements of such systems. Future realizations of this or similar concepts will show how to achieve end user acceptance and distributors confidence in handling privacy and safety relevant features of their systems.

Acknowledgment

The authors thank all members of project “Smart Geothermal Energy Grid Ruhr - GeoSmaGriR“ and other members of “ruhrvalley“ for their dedicated work. We thank A. Sekulski and A. Sinnaeve from the *Institute for Internet Security* for their great programming, support and comments.

References

[1] Gartner, “Gartner Says Worldwide IoT Security Spending Will Reach $1.5 Billion in 2018, “ 2018. [Online]. Available: https://www.gartner.com/newsroom/id/3869181 [Accessed: Aug. 09.2018]

[2] I. Stojmenovic, “Machine-to-Machine Communications With In-Network Data Aggregation, Processing, and Actuation for Large-Scale Cyber-Physical Systems,“ in IEEE Internet of Things Journal, vol. 1, no. 2, pp. 122-128, April 2014.

[3] M. M. Hossain, M. Fotouhi and R. Hasan, “Towards an Analysis of Security Issues, Challenges, and Open Problems in the Internet of Things, “ 2015 IEEE World Congress on Services, New York, 2015, pp. 21-28.

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

[5] Trusted Computing Group, “TPM 2.0 Library Specification, “ 2016. [Online]. Available: https://trustedcomputinggroup.org/resource/tpm-library-specification/ [Accessed: Aug. 01, 2018]

[6] S. Wendzel, W. Mazurczyk and G. Haas, “Dont You Touch My Nuts: Information Hiding in Cyber Physical Systems, “ 2017 IEEE Security and Privacy Workshops (SPW), San Jose, CA, 2017, pp. 29-34.

[7] K. Wan and V. Alagar, “Achieving Dependability of Cyber Physical Systems with Autonomic Covering,“ 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing, Dalian, 2014, pp. 139-145.

[8] The OWASP Foundation, “Top IoT Vulnerabilities, “ 2016. [Online]. Available: https://www.owasp.org/index.php/Top_IoT_Vulnerabilities [Accessed: Sep. 10.2018]

[9] S. Emerson, Y. Choi, D. Hwang, K. Kim and K. Kim, “An OAuth based authentication mechanism for IoT networks, “2015 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, 2015, pp. 1072-1074.

[10] S. Cirani, M. Picone, P. Gonizzi, L. Veltri and G. Ferrari, “IoT-OAS: An OAuth-Based Authorization Service Architecture for Secure Services in IoT Scenarios,“ in IEEE Sensors Journal, vol. 15, no. 2, pp. 1224-1234, Feb. 2015.

[11] M. Thumann, D. Rcher, “NAC@ACK: Hacking the Cisco NAC Framework,“ BlackHat Conference, March 2007.

[12] M. Jeong Park, D. Kwan Kim, W. Kim and S. Park, “Dynamic Software Updates in Cyber-Physical Systems,“ 2010 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, 2010, pp. 425-426.

[13] S. Kang, I. Chun and W. Kim, “Dynamic software updating for cyber-physical systems,“ The 18th IEEE International Symposium on Consumer Electronics (ISCE 2014), JeJu Island, 2014, pp. 1-3.

[14] Arthur, W., Challener, D. and Goldman, K. (2015). A Practical Guide to TPM 2.0. Berkeley, CA: Apress.

[15] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3“, RFC 8446, 2018, [Online]. Available: https://tools.ietf.org/html/rfc8446 [Accessed: Sep. 12.2018]

[16] S. Han, W. Shin, J. Park, and H. Kim, “A Bad Dream: Subverting Trusted Platform Module While You Are Sleeping,“ Proceedings of the 27th USENIX Security Symposium, USENIX, pp. 1229-1246, 2018

[17] Federal Office for Information Security in Germany (BSI), “The State of IT Security in Germany 2017, “ 2017. [Online]. Available: https://www.bsi.bund.de/EN/Publications/SecuritySituation/SecuritySituation_node.html [Accessed: Sep. 10.2018]

[18] Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen, “Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung, “ 2016 [Online]. Available: https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/ Sachgebiete/QES/Veroeffentlichungen/Algorithmen/2016Algorithmenkatalog.pdf [Accessed: Sep. 10.2018]