A Secure Platform Model Based on ARM Platform Security Architecture for IoT Devices

Junyoung Jung, Beomseok Kim, Jinsung Cho, and Ben Lee, *Member, IEEE*

***Abstract*—The proliferation of Internet of Things (IoT) devices comes with many challenges among which security is one of the most serious issues. In order to address the security issue for low-end IoT devices, ARM recently proposed the platform security architecture (PSA), which provides execution isolation to safely manage and protect the computing resources of low-end IoT devices. However, developers implementing IoT services for PSA-based IoT devices need to follow complex development procedures and understand the PSA hardware, which dramatically increases the development time and cost of PSA-based IoT devices. This article analyzes vulnerabilities that may arise from general-purpose low-end IoT devices to derive the security requirements and essential security services for PSA-based IoT devices, and proposes a secure platform model based on the analysis results. The proposed secure platform model consists of System Security Services and Application Security Services based on the basic PSA model and essential trusted subsystems, and it is designed to be flexible and applicable to various types of PSA-based IoT devices. In addition, it provides secure platform services APIs to enable easy and fast development of IoT services. To evaluate the proposed secure platform model, two proof-of-concept implementations are provided by using both the basic PSA model with secure element (SE) and a reference device for ARM's PSA. Finally, a case study shows that the development of IoT services can be done easily and quickly using the proposed security platform model.**

***Index Terms*—ARM platform security architecture (PSA), Internet of Things (IoT) security, security platform, security service, trusted execution environment (TEE).**

I. Introduction

With the rapid growth of Internet of Things (IoT), the number of IoT devices has surpassed the number of mobile phones in 2018 and is expected to reach 18 billion devices by 2022 [1]. This growth is driven by a wide range of services, including home automation (e.g., smart locks and thermostats), industrial automation, connected vehicles, smart cities, water management, agriculture, and even vending machines [2]. Consequently, IoT has also become an attractive target for attackers and well-known threats vary from simple resource hijacking, such as Mirai and Hajime botnets [3], [4], to IoT ransomware targeting critical infrastructures such as power grids [5].

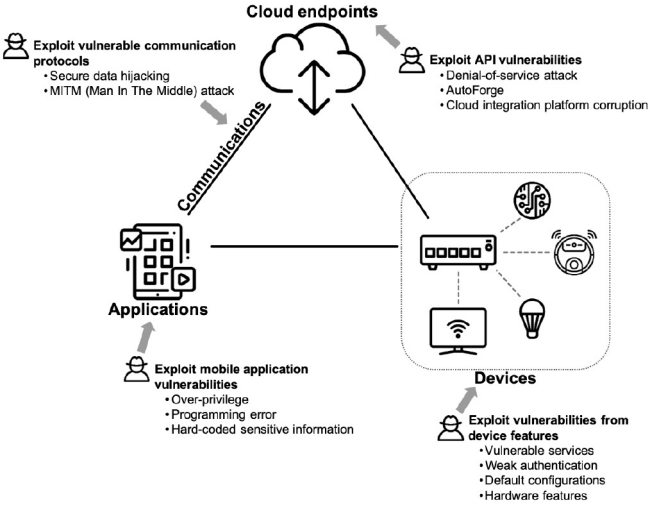

Fig. 1. Vulnerabilities and attacks in the entities that comprise the IoT systems.

Typically, IoT systems are composed of cloud servers, applications, and devices, and they are connected to each other as shown in Fig. 1. IoT security issues can arise in various ways for each entity in the system. First, cloud endpoints are Internet components for IoT services, and they provide core services such as remote administration, alerts, and digital contents [6]. Various cloud APIs exist for the development of such cloud endpoints, but they can be vulnerable. For example, attackers can inject inappropriate footage, trigger false alarms, and launch Denial-of-Service (DoS) attacks to the cloud endpoints through these insecure APIs [7], [8]. In addition, an access token hijacking may occur using forged requests from applications to cloud endpoints [9]. Second, applications are typically implemented as mobile applications and they are vulnerable to threats, such as over-privileged access, programming error, and hard-coded sensitive data [10]–[12]. Finally, devices that represent other endpoints of IoT systems also have vulnerabilities, such as software bugs, weak authentication, insecure default configuration, and hardware susceptibility (e.g., easily extractable flash memory, slow cryptographic functions, easily accessible debug interface, etc.), and there are many

Manuscript received April 20, 2021; revised June 1, 2021; accepted August 26, 2021. Date of publication September 1, 2021; date of current version March 24, 2022. This work was supported in part by the Young Researcher Support Program through NRF Grant founded by the Ministry of Science, ICT & Future Planning (MSIP) under Grant NRF-2018R1C1B6006938, and in part by the Basic Science Research Program through NRF Grant funded by the Ministry of Education under Grant NRF-2017R1D1A1B04035914. *(Corresponding author: Beomseok Kim.)*

Junyoung Jung, Beomseok Kim, and Jinsung Cho are with the Department of Computer Engineering, Kyung Hee University, Yongin-si 17104, South Korea (e-mail: junyoung.jung@khu.ac.kr; passion0822@khu.ac.kr; chojs@khu.ac.kr).

Ben Lee is with the School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331 USA (e-mail: benl@eecs.orst.edu).

Digital Object Identifier 10.1109/JIOT.2021.3109299

attacks that exploit these vulnerabilities [13]–[15]. Meanwhile, the infrastructure connecting IoT components consists of various communication technologies, such as WLAN, ZigBee, long range (LoRa), and Bluetooth low energy (BLE). Since these are generally implemented using lightweight protocols, they are vulnerable to various attacks, such as hijacking and man in the middle (MITM) attacks [16], [17]. Fortunately, many studies exist on dealing with vulnerabilities of cloud endpoints, applications, and infrastructures, and the aforementioned security problems can be handled with existing security techniques.

Unlike other entities in the system, IoT devices are generally implemented using low-end hardware platforms [18]. As such, they are susceptible to various vulnerabilities, such as sensitive information leak, firmware modification attack, firmware roll-back attack, etc., because bootloader/firmware binaries are stored in easily accessible storage and can be analyzed and modified [19], [20].

To deal with these problems, various research on enhancing the security of low-end devices have been performed, and some of them are implemented as software-based security services [21], [22]. However, software security implementations may cause additional vulnerabilities [23]. In addition, security and privacy concerns for IoT devices are particularly difficult due to the short time-to-market (TTM) [24]. Despite these problems, studies on systems that provide hardware Root of Trust (RoT)-based integrated security support for low-end IoT devices are still in their early stage. The trusted computing group (TCG) defines RoT as a component that performs one or more security-specific functions, such as measurement, storage, reporting, verification, and/or update. An RoT is always trusted to behave in an expected manner, and studies have shown the importance of using hardware RoT (i.e., nonmodifiable boot ROM, cryptographic processor, etc.) to improve security in low-end devices [26], [27].

Meanwhile, ARM recently proposed the platform security architecture (PSA) as a security solution for low-end IoT devices based on Cortex-M series processors [28]. The ARM PSA has two execution states to provide a trusted execution environment (TEE), a secure state (Secure World) and a nonsecure state (Normal world), and execution isolation between them. Accordingly, application functions are divided into two parts, i.e., secure processing environment (SPE) firmware and non-SPE (NSPE) firmware, in the design phase and developed separately by Secure World developers (SWDs) and normal world developers (NWDs), respectively. However, the ARM documentation only specifies the minimum required components and functions to implement the PSA. The requirements for adding additional hardware RoT elements to address vulnerabilities caused by software bugs, such as a trusted subsystem, are not defined and instead delegated to platform providers. Therefore, NWDs and SWDs would have to be familiar with both the PSA as well as trusted subsystem hardware, including switching between secure and nonsecure states. Since applications for low-end IoT devices are quite simple, this complicated development process will increase the development time and cost. The development time and cost can be significantly reduced if hardware developers and SWDs can provide a secure platform so that NWDs can develop IoT applications without the knowledge of the complicated PSA hardware.

Based on the above-mentioned discussions, this article proposes a model that includes a secure software platform and essential trusted subsystems based on the ARM PSA hardware platform. In order to derive the security requirements for the proposed model, a comprehensive analysis is first performed on vulnerabilities and threats by considering hardware restrictions of low-end devices and the IoT standard document on security requirements. Based on the derived security requirements, the proposed secure platform model includes five security services based on a hardware design that includes identified essential trusted subsystem and secure platform APIs for NWDs. The proposed secure platform model is a general model that not only provides guidelines for hardware developers and SWDs to develop a PSA-based hardware platform, but also helps NWDs to quickly develop nonsecure firmware. Our evaluation of two Proof-of-Concept (PoC) implementations shows that the proposed model is useful for the PSA-based platform development. In addition, our evaluation shows that application developers can easily and quickly develop secure IoT applications by demonstrating a smart plug device.

The remainder of this article is organized as follows: Section II presents a preliminary study as well as a background on ARM TrustZone and PSA. Section III analyzes vulnerabilities, threats, and requirements of low-end IoT devices based on the oneM2M standard and ARM's PSA documents. The proposed security platform model is presented in Section IV. Section V discusses our PoC implementations of the proposed security platform model. Section VI presents a case study on the development of a secure IoT device based on the secure platform model. Finally, Section VII concludes this article and discusses possible future work.

II. Preliminary Study and Background

A. Preliminary Study

As a preliminary study, firmware from 16 commercial IoT devices were extracted and analyzed[1] to determine whether commercial IoT devices employ security solutions based on the hardware RoT support and how easily their firmware can be accessed. This was achieved by desoldering the flash memory chips on each device and obtaining the firmware binaries using a flash memory reader. The extracted firmware files were then analyzed using various analysis tools, such as Binwalk, IDA pro, Ghidra, etc.

The results of our study can be summarized as follows: 1) the hardware platforms for most of the existing IoT devices are designed without security considerations,

[1]The 16 commercial IoT devices examined were as follows: Samsung Galaxy S9 (smart phone), Synology DS 218J (NAS), Xiaomi Mi Box S (set top box), LG VR6341LVM (smart TV), FOSCAM (IP camera), Xiami Mi Smart Home Gateway 2 (gateway), Broadlink RM-MINI3 (gateway), ipTime A1004ns (router), Gateman DANDY S (doorlock), DJI Tello (drone), SYMA X8W (drone), Brunt BREAKR1601 (blind engine), Xiaomi Mijia Bluetooth Smart Temperature & Humidity Sensor (smart sensor), Xiaomi PM2.5 Sensor (smart sensor), SK BDS301W (smart switch), and Darwin PM-B540-W (smart meter).

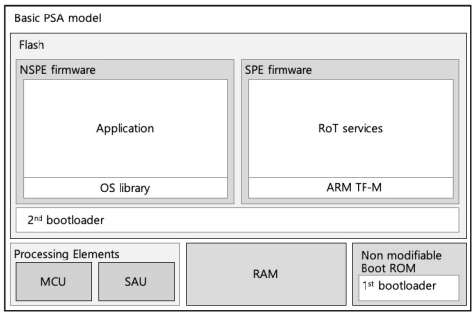

Fig. 2. Basic PSA model.

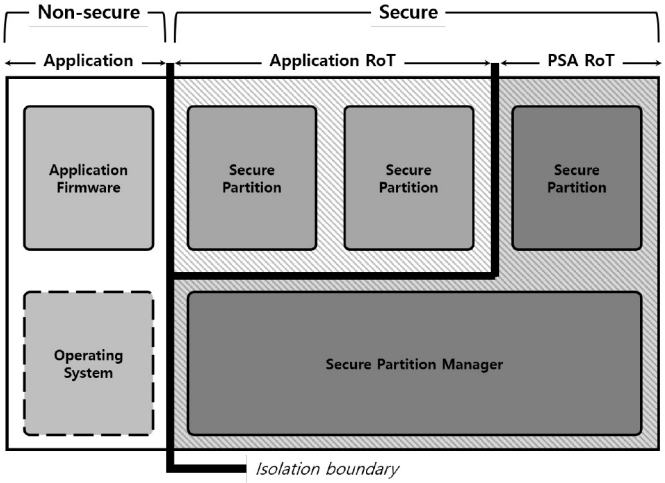

Fig. 3. Elements of the PSA firmware framework.

and their firmware can easily be extracted; 2) reverse engineering the extracted firmware binaries, including disassembling/decompiling, makes it possible to obtain essential data, such as credentials and even the main flow of control of codes; and 3) high-end Linux devices are much more vulnerable than bare-metal devices, and for this reason, Samsung Galaxy and LG smart TVs rely on the ARM's TrustZone technology. The implication of this study is that low-end IoT devices are being developed without any security considerations, and thus this poses a tremendous threat to IoT services.

B. ARM Platform Security Architecture

TrustZone is a hardware platform that improves security by providing an execution isolation environment on the ARM Cortex-A processor. Based on this, the ARM PSA was recently proposed to provide TEE for low-cost IoT devices [28]. Fig. 2 illustrates the basic ARM PSA model consisting of the essential components, which includes processing elements (PEs), RAM, boot ROM, and flash memory. PEs consist of ARM Cortex-M v8 microcontroller (MCU) and security attribution unit (SAU), which supports system state transition in an isolated execution environment. Flash memory stores the PSA firmware framework that includes SPE firmware and NSPE firmware.

Fig. 3 shows the elements of ARM's PSA firmware framework that isolates the Secure World and the Nonsecure World. Both worlds have their own user along with cache, memory, and other resources. In addition, the Secure World is

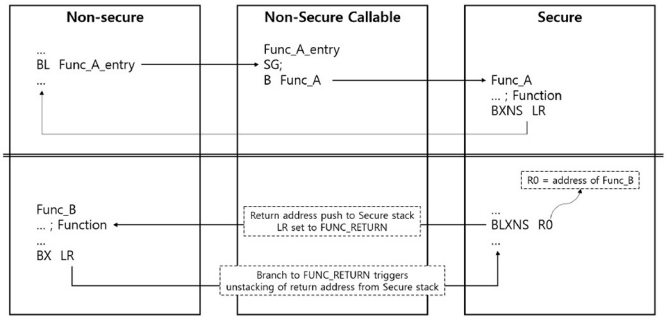

Fig. 4. System state transition of PSA firmware framework.

subdivided into *Secure Partitions* according to their purpose and classified into *PSA RoT* and *Application RoT*. The security functions essential to the system are implemented in the PSA RoT, while the Application RoT includes security functions used by nonsecure application firmware. The *Secure Partition Manager*, which is the most privileged firmware, protects the PSA RoT by managing isolation among PSA RoT, Application RoT, and Application. In addition, it provides communications among the isolated firmware components.

Fig. 4 shows the two cases of system state transitions [32]. In the first case, the system state transitions from the Nonsecure state to Secure state occurs when a code in the nonsecure region calls a secure function in the secure region. During the call, the processor allocates a special type of secure location called nonsecure callable (NSC) to properly return from executing the secure function. NSC executes the Secure Gateway (`SG`) instruction to store an entry point. When the secure function completes, the state is switched to the nonsecure state by using the `BXNS` instruction to return to the entry point stored in the NSC. In the second case, the state transitions from the secure state to the nonsecure state occurs when a code in the secure region calls a nonsecure function in the nonsecure region. This is accomplished by executing the `BLXNS` instruction, which causes the return address in the link register (LR) to be set to a special value called `FNC_RETURN` to switch to the secure state after the execution of the nonsecure state is completed. These system state switches are performed securely with the support of the SAU embedded in the ARM Cortex-M v8 processor.

ARM also defines the main components of the PSA RoT as follows.

1) *PSA Security Lifecycle:* Identification and control of available credentials of the device during the manufacturing stage of the PSA-based device.
2) *PSA Immutable RoT:* Nonmodifiable firmware and data provisioned during manufacturing.
3) *Trusted Boot and Firmware Update:* Ensures the integrity and authenticity of device firmware.
4) *Secure Partition Manager:* Manages the isolation of the RoT services and scheduling logic to service requests.
5) *PSA RoT Services:* Essential encryption functions and access management of immutable RoT for Application RoT services.

The main components of the PSA RoT are essential items that platform providers should implement when

TABLE I
VULNERABILITIES AND SECURITY REQUIREMENTS OF LOW-END IoT DEVICES

Vulnerabilities	Security requirements on oneM2M (refer to TS-0002 document)
Exposed Firmware Code	**SER-013**: The oneM2M System shall be able to provide the mechanism for integrity-checking on boot, periodically on run-time, and on software upgrades for software/hardware/firmware component(s) on M2M Device(s).
	SER-064: The M2M Devices shall provide a mechanism to prevent installation or modification of the software/middleware/firmware which run on the M2M Devices, unless it is authorized by an allowed stakeholder.
	SER-065: The oneM2M System shall be able to detect installation or modification of the software/middleware/firmware of M2M Devices that has not been authorized by an allowed stakeholder.
	SER-066: The oneM2M System shall enable allowed stakeholders to restrict or prevent operation of M2M devices using software/ middleware/firmware that the stakeholders did not authorize.
	SER-078: The oneM2M System shall be able to trigger the secure (e.g. authenticity, integrity, and confidentiality protected) Firmware/Software update of field devices.
Exposed Cryptographic Key	**SER-010**: The oneM2M System shall be able to support mechanisms for protection against misuse, cloning, substitution or theft of security credentials.
	SER-019: The oneM2M System shall be able to use service-level Credentials present inside the M2M Device for establishing the M2M Services and M2M Applications level security.
	SER020: The oneM2M System shall enable legitimate M2M platform providers to provision their own Credentials into the M2M Devices/Gateways.
	SER-021: The oneM2M System shall be able to remotely and securely provision M2M security Credentials in M2M Devices and/or M2M Gateways.
Exposed Sensitive Data	**SER002**: The oneM2M System shall be able to ensure the Confidentiality of data.
	SER003: The oneM2M System shall be able to ensure the Integrity of data.
	SER-026: The oneM2M System shall be able to provide mechanism for the protection of Confidentiality of the geographical location information
Software-based Cryptographic Function	**SER-068**: The information exchanged within the oneM2M System shall use cryptographic technology to ensure information authentication and information integrity.
	SER-069: The oneM2M System shall be able to securely transfer information by using an appropriate method such as digital signature.

developing PSA-based IoT devices. A simple way to implement each component of the PSA RoT is to develop the required functions in software. However, software-based functions are vulnerable to firmware extraction and tampering.

To deal with this problem, the ARM PSA document recommends that platform providers use a trusted subsystem (see Section III-B), which includes cryptographic accelerator, true random number generator (TRNG), secure storage (e.g., one-time-programmable memory and secure flash), and other trusted peripherals. A trusted subsystem may itself implement its own secondary RoT, but it must be a subordinate to the PSA RoT service at all times in the sense that its configuration and state must always be attestable by the PSA RoT.

Meanwhile, ARM TrustZone, which provides TEEs for Cortex-A series processors, is actively used to enhance the security of smartphones [29]. TEEs are often assumed to be highly secure, however, TEEs have been attacked multiple times [30]. ARM is enhancing security by continuously supplementing the vulnerabilities of attacks. Likewise, the ARM PSA is a technology that inherits TrustZone and is the most advanced trustable technology that can guarantee the security of low-end IoT devices. Therefore, this article assumes that the ARM PSA is a trustable security solution.

Section III further analyzes the security requirements of IoT devices to define the set of components for the proposed secure platform based on the features of the ARM PSA hardware platform.

III. ANALYSIS OF SECURITY REQUIREMENTS FOR IoT DEVICES

A. Vulnerabilities of Low-End IoT Devices

The two major limitations of low-end IoT devices are insecure storage and low computing power. First, low-end devices store their firmware codes and data, such as credentials and sensitive data, on various types of storage, e.g., flash memory, EPROM, EEPROM, etc. As mentioned in Section I, firmware codes and data in these storage media can be easily obtained through various hardware debug interfaces or by de-soldering chips. Second, low-end devices cannot perform computational intensive encryption/decryption operations due to low computing power and limited memory size. Therefore, this article defines the following four major vulnerabilities[2] of low-end IoT devices: 1) exposed firmware code; 2) exposed cryptographic key; 3) exposed sensitive data; and 4) software-based cryptographic function.

On the other hand, oneM2M, which is the largest standardization organization for IoT/M2M services [33], published the Technical Specification—0002 document that includes security requirements for Machine-to-Machine (M2M) and IoT services, networks, and devices. The current version of the specification defines 82 security requirements (SER-001–082) [34]. These requirements were examined, and the security requirements related to system issues (not application ones) of IoT devices and their four major vulnerabilities defined above were identified. Table I summarizes the mapping.

[2]The application software vulnerabilities are beyond the scope of this article.

In addition, various hardware RoT support mentioned in Section I are presented as follows.

1) *SER-023:* The oneM2M System shall be able to rely on the hardware security module (HSM)[3] to provide local security, if it is supported.
2) *SER-070:* The oneM2M System shall be able to support security mechanisms to protect cryptographic keys and operations by using tamper-resistant elements, such as trusted platform module (TPM), HSM, and subscriber identity module (SIM).
3) *SER-042:* A secured API shall enable application and service layer entities to make use of sensitive functions and data residing within the secure environment, independently of its technical implementation.

In SER-042, a secured API is clearly described, and this is the major function utilized by the proposed secure platform model.

Based on Table I, the following analyzes each vulnerability in more detail along with related security requirements and presents some countermeasures.

1) Exposed Firmware Code: As discussed in Section III-A, firmware in IoT devices can easily be extracted, and there are a number of recent studies on firmware extraction/analysis/modification. Zhang *et al.* [35] developed a binary firmware extraction framework called PANDORA and used it to perform security analysis of firmware binaries. The purpose of PANDORA is to show that an attacker can analyze firmware codes of low-end IoT devices and even attempt to tamper with them. Vasile *et al.* [36] also conducted a study on tampering with firmware from actual off-the-shelf IoT devices. They found that firmware installed on a device can be extracted through the debug interfaces, such as JTAG and UART, as well as using raw flash dumps. To verify this, they performed several case studies on various commercial low-end IoT devices, such as Amazon fire TV, Google Nest, Chromecast, LG smart refrigerator, Samsung smart doorlock, etc. Although they extracted firmware using debug interfaces, our preliminary study reveals that most recent commercial IoT devices no longer provide the debug interfaces

To avoid vulnerabilities exposed by firmware codes, a mechanism to verify the integrity of a firmware during boot and update phases need to be provided. In addition, IoT devices should be able to report their current operation status to a remote verifier to perform an integrity check of running firmware binaries. Furthermore, these mechanisms should be supported based on a hardware RoT to satisfy the security requirements SER-013, SER-064, SER-065, SER-066, and SER-078 in Table I.

2) Exposed Cryptographic Key: A cryptographic key is an essential component for a variety of security operations, such as authentication and digital signature. As mentioned in Section I, exposed keys can be illegally copied or replaced with other keys, and this can significantly compromise the reliability of IoT services. Strobel *et al.* [37] presented several key extraction cases of low-end devices and exposed their vulnerability. They extracted system keys by performing hardware analysis on Microchip PIC16F886μC connected to SimonsVoss digital locking system and YubiKey one-time password token.

A countermeasure against these threats should be provided by storing cryptographic keys in a secure storage based on a hardware RoT support, which will then satisfy requirements SER-010, SER-019, SER-020, and SER-021 in Table I.

3) Exposed Sensitive Data: IoT devices store confidential data, including configuration data, privacy data, etc., which can be exposed. The aforementioned work by Vasile *et al.* [36] also mentioned the vulnerability of exposed sensitive data. In this study, the authors achieved root privileges on 18 off-the-shelf IoT devices and were able to easily access security-sensitive data.

To prevent this threat and to satisfy the requirements SER-002, SER-003, and SER-026 in Table I, a Data Protection mechanism needs to be provided. This will allow sensitive data to be stored in an encrypted form, which is safely provisioned, as discussed in the previous section. For example, data can be stored or encrypted securely with a unique key in a secure storage.

4) Software-Based Cryptographic Function: Software-based cryptographic functions are also exposed to attackers, as mentioned in Section III-A1. Sklavos *et al.* [38] analyzed the limitations of software-based cryptographic functions on general-purpose computers. According to their analysis, software-based cryptographic functions are vulnerable to attacks, such as tamper and injection, which can cause the leakage of cryptographic keys and other sensitive data. Heartbleed, which is a bug in the OpenSSL cryptographic library widely used in the implementation of transport layer security (TLS), is a typical example of vulnerability in cryptography implementation [39]. This allows credentials, such as session keys and other secret information (e.g., user ID and password) to be obtained from SSL/TLS. Moreover, software cryptographic functions are not suitable for low-end devices due to their computational limitations.

To deal with these problems and to satisfy the requirements SER-068 and SER-069, a hardware implementation of the cryptographic algorithm (e.g., cryptographic accelerator) is required.

B. Comprehensive Security Analysis of PSA-Based Low-End IoT Devices

One way to protect against the vulnerabilities discussed in the previous section is to utilize the PSA to ensure isolated execution. However, platform providers need to understand the design of the PSA hardware platform including an appropriate trusted subsystem. In addition, a firmware that is divided into SPE firmware and NSPE firmware should be developed by SWDs and NWDs, respectively, and as such all developers need to be familiar with the PSA and trusted hardware subsystems. This complex development process increases the cost and time of IoT device development.

In order to design PSA-based low-end IoT devices, this article regards trusted subsystems as essential for implementing

[3]HSM is a term defined by oneM2M and is synonymous with the trusted subsystem of the ARM PSA.

TABLE II
SECURITY SERVICES TO SATISFY REQUIREMENTS TO IMPLEMENT PSA-BASED LOW-END IoT DEVICES

Vulnerabilities	Security requirements	Related main components of PSA RoT	Required trusted sub-systems			Security services	Service group
			Non-modifiable boot ROM	Secure key storage	Cryptographic accelerator & TRNG		
Exposed firmware code	SER-013 SER-066	PSA security lifecycle PSA immutable RoT Trusted boot and firmware update	•	•	•	Secure boot	System security service
	SER-013 SER-064 SER-078	PSA security lifecycle PSA immutable RoT Trusted boot and firmware update		•	•	Secure firmware update	
	SER-013 SER-064 SER-065	PSA security lifecycle PSA immutable RoT		•	•	Attestation	Application security service
Exposed sensitive data	SER-002 SER-003 SER-026	PSA security lifecycle PSA RoT services		•	•	Data protection	
Exposed cryptographic key	SER-010 SER-019 SER-020 SER-021	PSA security lifecycle PSA RoT services		•	•	Data protection Key management	
Software-based cryptographic functions	SER-068 SER-069	PSA security lifecycle PSA RoT services		•	•	Cryptographic functions	

the main components of the PSA RoT. In addition, well-known security services directly related to vulnerabilities, security requirements, and the implementation of the main components of the PSA RoT are identified in Table II. In order to solve the exposed firmware code vulnerability and implement the related main components of the PSA RoT, services that verify the integrity of the system at boot and runtime and ensure the latest verified firmware updates are essential. Exposed sensitive data, exposed cryptographic key, and software-based cryptographic functions are vulnerabilities related to the execution of cryptographic functions. Therefore, services that can guarantee the stability of key management and encryption function execution are necessary to protect against these vulnerabilities and implement the related main PSA RoT components. In addition, in order to implement the main components of PSA RoT, it is necessary to implement essential security services based on the support of trusted subsystems, such as unmodifiable boot ROM, security key storage, password accelerator, and TRNG.

As a result, the five essential security services needed to implement the main components of the PSA RoT are as follows: 1) Secure Boot; 2) secure firmware update;[4] 3) Data Protection; 4) Key Management And Cryptographic Functions; and 5) Attestation. SWDs can implement these five security functions as a secure platform, and a trusted subsystem and the normal world can use these service APIs without the knowledge of complicated PSA and trusted subsystem hardware. Details of our secure platform based on the ARM PSA that will lead to fast TTM and reduced cost are discussed in the following section.

IV. PROPOSED SECURE PLATFORM

This section presents the implementation of the proposed secure platform, which includes trusted subsystems, securely provisioning credentials, and providing secure services to satisfy the requirements in Table II.

[4]Secure boot and secure firmware update are the same concepts as *trusted boot* and *firmware update* of the ARM PSA.

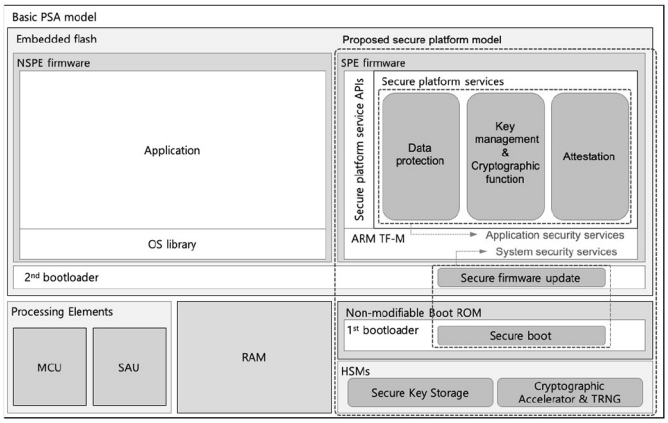

Fig. 5. Proposed secure platform model.

A. Secure Platform Model

Fig. 5 shows the proposed secure platform model based on the ARM Cortex-M v8 architecture. As mentioned in Section II-B, the basic ARM Cortex-M v8 architecture consists of PE, RAM, and embedded flash. Based on this architecture, the proposed secure platform model includes essential trusted subsystems, secure platform services, and secure platform services APIs, as shown on the right side of Fig. 5. As discussed in Section III, Trusted Subsystems, such as nonmodifiable boot ROM, Secure Key Storage, cryptographic accelerator, and TRNG are necessary to satisfy the requirements defined by the oneM2M standard.

The secure platform services listed in Table II can be categorized into two groups according to their purpose: 1) System Security Services and 2) Application Security Services. *System Security Services* composed of Secure Boot and Secure Firmware Update are the basic services to ensure system

TABLE III
PLATFORM KEYS AND THEIR USAGE

Platform keys	Description	Security services
$K^{PP}_{private}$	Private key of platform provider	Secure Boot
K^{PP}_{public}	Public key of platform provider	
$K^{AD}_{private}$	Private key of application developer	Secure Boot, Secure Firmware Update
K^{AD}_{public}	Public key of application developer	
$K^{Device}_{private}$	Private key of device	Attestation
K^{Device}_{public}	Public key of device	
DUK	Device Unique Key	Data Protection

integrity, confidentiality, and availability. *Application Security Services* consisting of Data Protection, key management, and cryptographic functions, and Attestation are essential services for NWDs to develop NSPE firmware in a Nonsecure World. According to the isolation requirement of the ARM PSA, these secure platform services have to be securely implemented in a Secure World because they are executed through system state transition using `NSC` and `SG` instructions.

B. Platform Keys for Secure Platform Services

In the proposed secure platform, the secure platform services guarantee integrity, confidentiality, and availability through various cryptographic algorithms using *platform keys* shown in Table III. These keys are stored in the *Secure Key Storage* during the initial provisioning stage. The usage of platform keys for secure platform services are as follows.

1) Secure Boot is a chain of trust, where the first bootloader verifies the second bootloader and the second bootloader verifies the SPE firmware and NSPE firmware. The detailed use of platform keys during executions of Secure Boot are as follows.
   a) The first bootloader verifies the digital signature of the second bootloader using K^{PP}_{public}.
      i) The platform provider signs the second bootloader with K^{PP}_{private} and stores this signature together with the second bootloader during the initial provisioning or firmware update.
   b) The second bootloader verifies both digital signatures of the SPE firmware and the NSPE firmware using K^{PP}_{public} and K^{AD}_{public}.
      i) The platform provider signs the SPE firmware with K^{PP}_{private} and stores this signature along with the SPE firmware during the initial provisioning or firmware update.
      ii) The application developer signs the NSPE firmware with K^{AD}_{private} and stores this signature along with the NSPE firmware during the initial provisioning or firmware update.
2) The Secure Firmware Update service updates the device with the latest NSPE firmware. The detailed use of platform keys during the executions of the Secure Firmware Update service is as follows.
   a) The application developer creates an update package containing the new application firmware and its digital signature signed with K^{AD}_{private}.
   b) The firmware updater sends the update package to the Secure Firmware Update service on the device.
   c) The Secure Firmware Update service verifies the integrity of received package with K^{AD}_{public} and updates the new firmware.
3) The Data Protection service allows application developers to save data securely without a key and cryptographic functions by using *DUK*.
4) The Key Management And Cryptographic Function service provides functions to derive application dependent keys with *DUK* as well as manage and delete derived keys. Based on the derived keys, various cryptographic functions, such as SHA, RSA, and AES are also provided.
5) The attestation service proves that the system and application software are intact and trustworthy by using $K^{\text{Device}}_{\text{public}}$ and $K^{\text{Device}}_{\text{public}}$.
   a) To check the integrity of the system at runtime, the attestation service signs the attestation data with $K^{\text{Device}}_{\text{private}}$.
   b) The verifier checks the integrity of the measurement with $K^{\text{Device}}_{\text{public}}$.

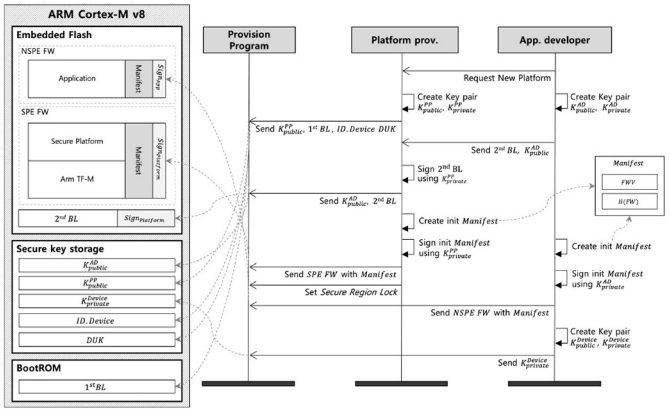

Fig. 6. Provisioning process.

In order to securely store and operate on sensitive data such as encryption keys, the proposed secure platform model undergoes a provisioning process of injecting data into a device during manufacturing as shown in Fig. 6. When a platform is designed, both SWD and NWD create their own key pairs. The SWD stores the generated K^{PP}_{public}, first bootloader (*1st BL*), device ID (*ID.Device*), and *DUK* to the device by using the provisioning tool. The second bootloader (*2nd BL*) received from the NWD is signed with K^{AD}_{private} and saved on the device with K^{AD}_{public}. Afterward, the SWD creates a manifest of the SPE firmware, signs it with K^{PP}_{private}, stores it on the device, and places a Secure Region Lock to isolate the SPE firmware. The NWD also creates a manifest of NSPE firmware, signs it with K^{AD}_{private}, and stores it on the device. Finally, the NWD creates a key pair for device verification, stores $K^{\text{Device}}_{\text{public}}$ in the device, and terminates device provisioning.

C. Secure Platform Services APIs

The following provides detailed descriptions of the APIs for Application Security Services.

1) Data Protection: As mentioned in Section IV-B, the Data Protection service ensures the confidentiality and integrity of various data, such as credentials, security-sensitive data, privacy information, etc. The following shows the sample APIs for the Data Protection service.

1) `int storeData(int *dataID, char *plainData, int dataSize);`
   a) *Input:* plain data and its size.
   b) *Output:* ID of encrypted data in `dataID`.
2) `int loadData(int dataID, char *plainData, int *plainSize);`
   a) *Input:* ID of encrypted data.
   b) *Output:* Decrypted data stored in `plainData` and its size in `plainSize`.

This service satisfies the requirements SER-002, SER-003, and SER-026 in Table II.

2) Key Management and Cryptographic Function: NWDs are required to use cryptographic functions for their applications as well as the Data Protection service. In addition, the keys for these cryptographic functions need to be managed. To satisfy these requirements, the proposed secure platform model provides an index-based key management where the value of the application key is not exposed to NWDs. This ensures that keys stored in the Secure Key Storage or using the Data Protection service is not exposed.

Examples of cryptographic APIs using the index-based key management are shown below, and different options exist depending on specific cryptographic algorithms.

1) `int createKey(int *keyID, cryptOpt option);`
   a) *Input:* Cryptographic option.
   b) *Output:* ID of the created key in `keyID`.
   c) `cryptOpt` is a structure that contains options for various cryptographic algorithms, such as AES, RSA, ECC, etc.
2) `int destroyKey(int keyID);`
   a) *Input:* ID of the key.
   b) *Output:* Destroy the key of `keyID`.
3) `int saveKey(int *keyID, keyOpt key, int keySize, cryptOption option);`
   a) *Input:* Key and its size with cryptographic option.
   b) *Output:* ID of saved key in `keyID`.
4) `int restoreKey(int keyID, keyOpt *key, int *keySize);`
   a) *Input:* ID of the key.
   b) *Output:* Key of the `keyID` in `key` and its size in `keySize`.

The requirements SER-010, SER-019, SER-020, and SER-021 in Table II are met by the above key related APIs.

The following shows APIs related to cryptographic functions.

1) `int hash(hashOpt option, char *plainData, int plainSize, char *hashData, int *hashSize);`
   a) *Input:* Hash option, plain data, and its size.
   b) *Output:* Hashed data stored in `hashData` and its size in `hashSize`.
   c) `cryptOpt` is a structure that contains options for various hash algorithms, such as SHA, MD5, CRC, etc.
2) `int enc(cryptOpt option, int keyID, char *plainData, int plainSize, char *cipherData, int *cipherSize);`
   a) *Input:* Cryptographic option, ID of the key, plain data, and its size.
   b) *Output:* Encrypted data in `cipherData` and its size in `cipherSize`.
3) `int dec(cryptOpt option, int keyID, char *cipherData, int cipherSize, char *plainData, int *plainSize);`
   a) *Input:* Cryptographic option, ID of the key, cipher data, and its size.
   b) *Output:* Decrypted data in `plainData` and its size in `plainSize`.

In addition, these cryptographic-related APIs are performed in the cryptographic accelerator, which satisfies the requirements SER-068 and SER-069 in Table II.

3) Attestation: The goal of attestation is to prove that SPE firmware and NSPE firmware are intact and trustworthy. The verifier (e.g., the remote server) trusts that attestation data are accurate by checking its integrity measurement, which may contain SPE firmware and NSPE firmware as well as application-specific data. Therefore, our proposed secure platform model provides the attestation service using the following API.

1) `int getSignedMeasure(appSpecific data, char *signedMeasure, int *signatureSize);`
   a) *Input:* Application-specific data which is added to SPE firmware and NSPE firmware.
   b) *Output:* Signed measure in `signedMeasure` and its size in `signatureSize`.

The attestation service satisfies the requirements SER-013, SER-064, and SER-065 in Table II.

V. PoC Implementation

This section presents the PoC implementation as well as our analysis of the secure platform model proposed in Section IV. Platform providers develop IoT devices by adding various configurations based on the basic PSA model according to their purpose. Therefore, the hardware platform of PSA-based IoT devices can be developed by either adding external components to the basic PSA model or using an SoC. To evaluate different hardware platforms, our PoC implementations involve two approaches: 1) a basic PSA model with secure element (SE) and 2) a PSA reference device. The NuMaker-PFM-M2351 development board from Nuvoton is used for both approaches [31]. This board includes the M2351KIAAE SoC chip based on the ARM Cortex-M23 processor with TrustZone-based isolation and is capable supporting PSA. In addition, M2351KIAAE is connected to a Secure Serial Flash Memory (W77F32W) chip [40] and a Wi-Fi module (ESP8266) [41] via SPI and UART, respectively. The following sections provide details of the implementations.

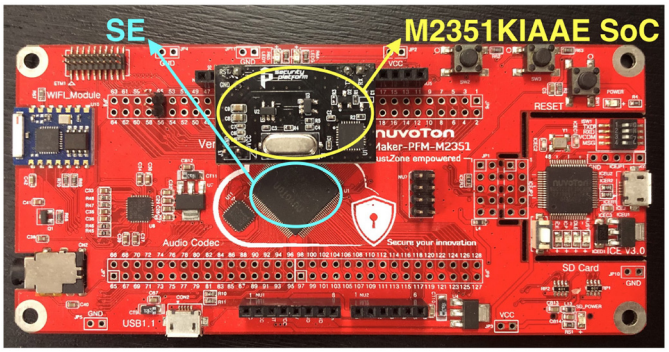

Fig. 7. PoC implementation: basic PSA model with SE.

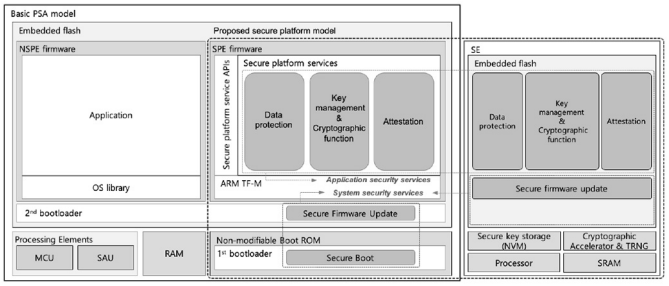

Fig. 8. System architecture of PoC implementation based on basic PSA model with SE.

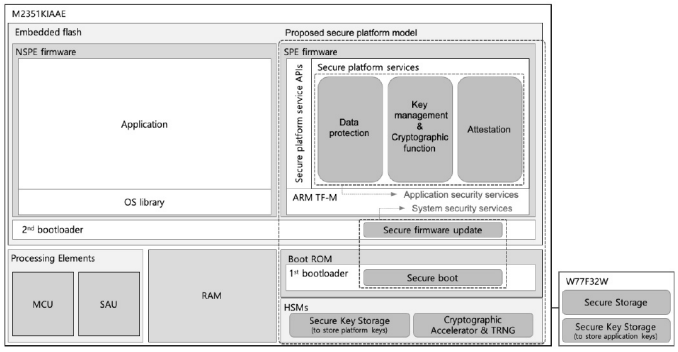

Fig. 9. System architecture of PoC implementation based on reference board for the ARM PSA.

A. Implementation 1: Basic ARM Cortex-M v8 With SE

There are various SE models on the market for low-end devices that have received high-grade common criteria (CC) certifications. Among them, Infineon Optiga Trust *P* SE[5] was chosen for the first implementation. The Infineon SE has a cryptographic accelerator with TRNG, processor, SRAM, non-volatile memory (NVM), and embedded flash. In addition, since an SE can embed its own proprietary firmware, a more secure implementation is possible by fusing firmware for the proposed secure platform services into an SE.

Figs. 7 and 8 show our prototype of the basic PSA with SE and its system architecture, respectively. The basic ARM Cortex-M v8 based PSA and SE are securely connected by a serial communication with SSL/TLS. In addition, Fig. 8 illustrates the mapping of the proposed secure platform model to the first PoC implementation, where NuMaker-PFM-M2351 with SE was used and only its basic PSA model was activated. The detailed description of the mapping is as follows.

1) *Secure Boot:* Implemented with nonmodifiable boot ROM and Secure Firmware Update. The basic ARM Cortex-M v8 processor provides a nonmodifiable boot ROM as default. Therefore, the first bootloader in our prototype is implemented in the nonmodifiable boot ROM. The second bootloader first establishes a secure session with the SE, and then the SPE firmware verification and Secure Firmware Update are implemented in the flash memory built into the SE.

[5]The Infineon Optiga Trust P model used in this article has CC EAL 5+ (high) certification. (https://www.infineon.com/dgdl/Infineon-OPTIGA_Trust_P_SLJ52_ACA-PB-v02_16-EN.pdf?fileId=db3a30434521785c01452440162809d8).

2) *Secure Key Storage and Cryptographic Accelerator:* Since the platform keys in Table III can be stored safely in NVM of the SE (i.e., Secure Key Storage) and it has a cryptographic accelerator, both requirements of essential trusted subsystems are satisfied.
3) *Data Protection:* Application's requests for the Data Protection service are delivered to the SE via secure platform APIs, and data is safely encrypted/decrypted with $K_{\text{shared}}^{\text{Device}}$.
4) *Key Management and Cryptographic Functions:* Similar to the Data Protection service, application requests are delivered to the SE through the SPE firmware. Generated keys are safely stored in the Secure Key Storage, and all the cryptographic operations are executed by the Cryptographic Accelerator.
5) *Attestation:* The attestation service module in the SE processes application's requests using $K_{\text{private}}^{\text{Device}}$ stored in the Secure Key Storage.

B. Implementation 2: Reference Device for PSA

In the second PoC implementation, we used only the target reference board implemented in SoC by integrating ARM Cortex-M v8 and trusted subsystem into an MCU (M2351KIAAE SoC), not SE connecting. Fig. 9 illustrates its system architecture.

In this PoC implementation, all the modules of the NuMaker-PFM-M2351 development board including external peripherals (i.e., Secure Serial Flash Memory, and Wi-Fi module) are activated. The SoC has NVM, OTP memory, nonmodifiable boot ROM, and cryptographic accelerator with TRNG. This cryptographic accelerator can directly access the Secure Serial Flash Memory chip that stores encrypted data (Secure data), but the main processor does not directly access the W77F32W chip. The Wi-Fi module can communicate with the outside through network protocols, such as TCP, UDP, and SSL. The detailed description of the mapping is as follows.

1) *Secure Boot With Nonmodifiable Boot ROM and Secure Firmware Update:* The first bootloader is implemented in the nonmodifiable boot ROM and the second bootloader is implemented in NVM. This is consistent with the proposed secure platform model.
2) *Secure Key Storage and Cryptographic Accelerator:* OTP in the M2351KIAAE SoC is used for provisioning

platform keys as nonmodifiable storage, and the Secure Serial Flash Memory chip is used for storing application dependent keys and security-sensitive data. M2351KIAAE also includes cryptographic accelerators and TRNG; therefore, both requirements of essential trusted subsystems are satisfied.

3) *Data Protection, Key Management, and Cryptographic Functions, and Attestation:* The platform keys including $K_{\text{shared}}^{\text{Device}}$ are provisioned in OTP and the application dependent keys are stored in the Secure Serial Flash Memory chip. However, unlike the first approach, these Application Security Services are implemented in the secure state in the ARM PSA. When a request related to these services from an application is generated, SAU converts the system state to a secure state, and the request operation is executed by the cryptographic accelerator.

Meanwhile, our PoC implementation presents two possibilities for applying essential HSMs to improve the security of devices supporting the ARM PSA. The first is to connect essential HSMs to an ARM Cortex M-23 or M-33-based MCU that supports the ARM PSA, and the second is to use an SoC, which includes both ARM Cortex M-23 or M-33 and essential HSMs. Our PoC is implemented by applying the proposed secure platform model to the NuMaker-PFM-M2351 reference board for two possibilities, which shows that the proposed secure platform model can be flexibly applied to other PSA-enabled devices.

C. Performance Analysis

Since IoT devices generally operate on batteries, low power consumption is one of the most important requirements for IoT devices. However, since existing IoT devices are exposed to various vulnerabilities, there is a need for techniques that can improve the security of IoT devices and at the same time satisfy low power consumption. This section verifies that the proposed security platform model can meet low power consumption requirements. In addition, the effect of some overheads of ARM PSA on IoT devices are also analyzed based on the experimental results.

For the experiments, simple encryption/decryption of three different scenarios were performed with the NuMarker-PFM-M2351 board in the same way as our PoC implementation. The encryption algorithm used AES ECB with a 128-bits key and the plain text "HELLO WORLD!" (13 characters). The three different scenarios represent the proposed secure platform model, Non-PSA with cryptographic accelerator, and Non-PSA with mbed-Crypto Library [48].

Fig. 10 shows the results of our performance evaluation. When a cryptographic accelerator is used, the execution time differs by about 2.59 μs depending on whether or not PSA is applied. In the proposed secure platform model, the actual overhead of one system state transition between secure and nonsecure states is 1.295 μs because two system state transitions occur.

To determine the effect of the state transition overhead on the power consumption of PSA-based IoT devices, the instructions for configuring the overhead has been analyzed.

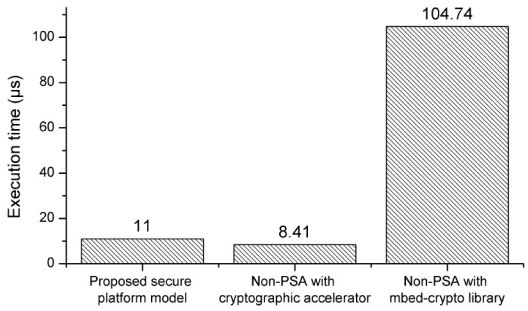

Fig. 10. Experimental results.

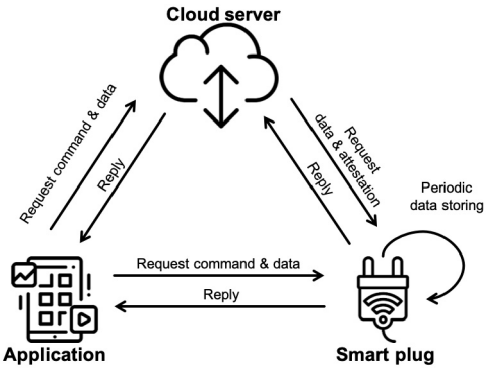

Fig. 11. Smart home/grid IoT service architecture with smart plug.

As mentioned in Section II, the state transition overheads consist of calling a function when entering NSC, calling an NSC callable function, calling a secure function, and initializing the general-purpose register (GPR)/process status register (PSR). Moreover, additional overheads occur in executing SG and BXNS instructions required to change the execution mode of the MCU. Therefore, the additional overheads that occur in the scenario with the PSA involves three jumps, a register initialization in the secure state, and an MCU execution mode change. Since this overheads comprise simple instructions, the PSA slightly increases power consumption compared to the scenario where the PSA is not supported.

On the other hand, the Non-PSA with mbed-Crypto Library scenario shows that using a cryptographic accelerator can reduce the computational power dramatically. The experimental results show that the scenario using the mbed-Crypto library takes about 12 times longer execution time than the scenario using the cryptographic accelerator, which means that using a software-based cryptographic function increases the load of the MCU. Such an increase in the load increases the power consumption and shortens the lifetime of IoT devices.

As a result, our performance evaluation verifies that the proposed security platform model based on the PSA can guarantee security with little overhead.

VI. Case Study

A smart plug is a power receptacle that plugs into a traditional electrical outlet and allows for integration into smart home/grid IoT services. The device periodically measures power consumption and reports it to the cloud server or the user application, as shown in Fig. 11. Our proposed secure platform model allows

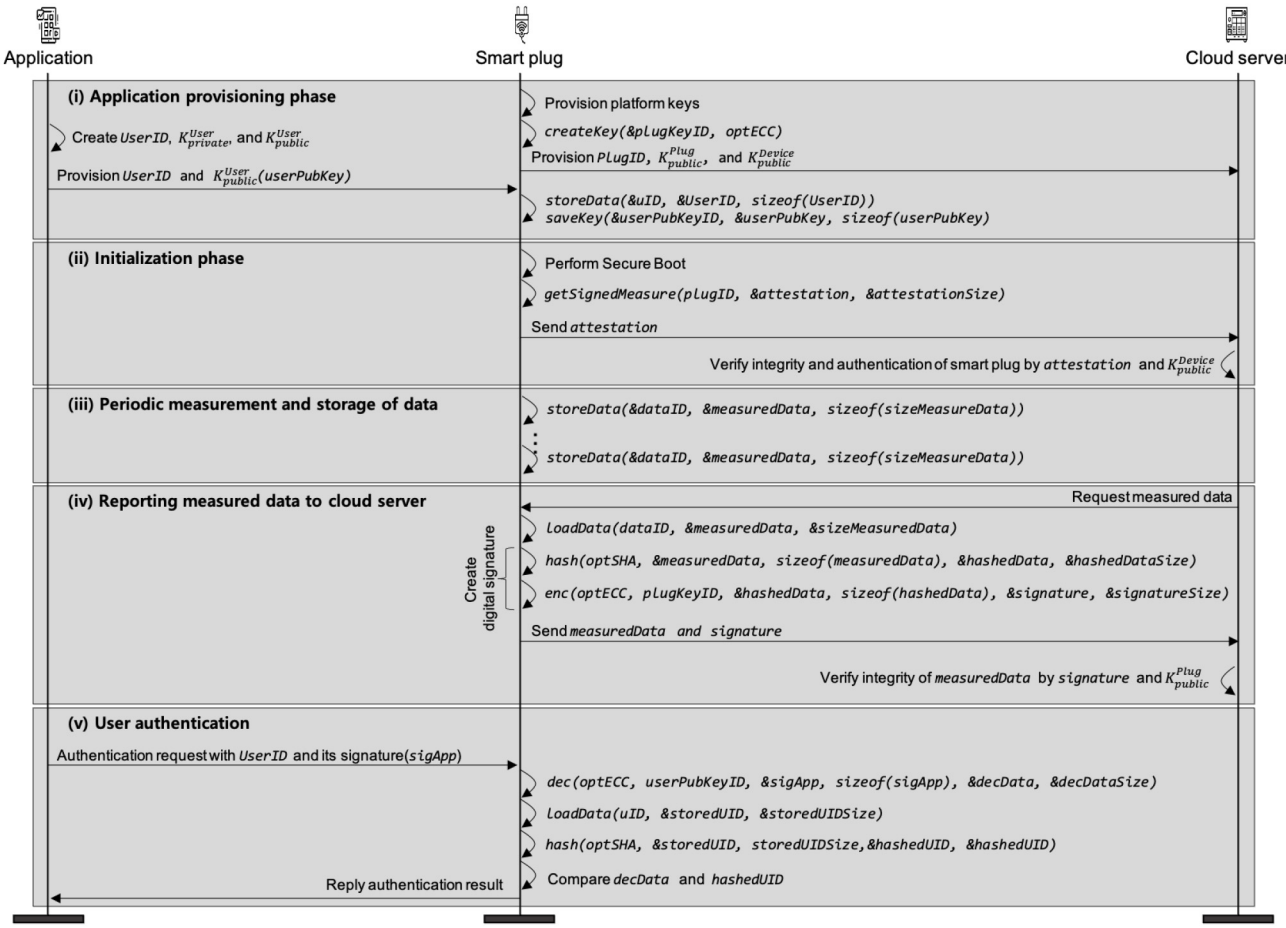

Fig. 12. Sequence diagram of smart plug.

an NWD to easily and quickly implement such a smart plug. Fig. 12 illustrates this simple scenario.

1) *Application Provisioning Phase:* First, the platform keys in Table III are provisioned in the smart plug. Next, the smart plug creates an ECC key pair with our Secure Service API `createKey()`. Finally, `UserID` and $K^{\text{User}}_{\text{public}}$ are safely stored using the APIs `storeData()` and `saveKey()`. This phase is only performed once.
2) *Initialization Phase:* When the smart plug is powered on or reset, the Secure Boot procedure starts and a remote attestation is performed using the API `getSignedMeasure()`. If the cloud server succeeds the integrity check of attestation data with $K^{\text{Plug}}_{\text{public}}$, the smart plug is authenticated. This attestation can also be done periodically or as needed.
3) *Periodic Measurement and Storage of Data:* The smart plug periodically measures the power consumption and stores this information using the API `storeData()`.
4) *Reporting Measured Data to Cloud Server:* When the cloud server sends a request, the smart plug prepares a reply message containing `measuredData` and `signature` using APIs `loadData()`, `hash()`, and `enc()`.
5) *User Authentication:* In order to control the smart plug with a smartphone application, the user needs to be first authenticated. To accomplish this, the application sends a message that contains `UserID` and its signature to the smart plug. The smart plug then authenticates the user with its `UserID` and $K^{\text{User}}_{\text{public}}$ provided during the application provisioning phase.

This case study shows that NWDs can easily develop IoT services that can meet security requirements using the secure platform services APIs.

VII. Conclusion

The demand for protection against vulnerabilities in low-end IoT devices is increasing with the proliferation of IoT. To solve this problem, ARM recently proposed PSA that provides a hardware-based isolation execution environment. However, the development cost and time of ARM PSA-based IoT devices dramatically increase because developers need to understand both the complex procedures of the PSA-based IoT devices development and the PSA hardware. To solve these problems, this article carried out a comprehensive analysis of vulnerabilities and security requirements for low-end IoT devices and derived essential trusted subsystems and security services to satisfy the security requirements. Based on this, we proposed a secure platform model for low-end IoT devices based on ARM PSA. The proposed secure platform model not only reduces the development cost and time for platform providers, but also helps NWDs to quickly and easily develop IoT applications for low-end IoT devices based on PSA.

REFERENCES

[1] Ericsson. (2018). *IoT Market Outlook*. [Online]. Available: https://www.ericsson.com/en/networks/trending/hot-topics/iot-connectivity/iot-market-outlook

[2] M. Xu *et al.*, "Dominance as a new trusted computing primitive for the Internet of Things," in *Proc. IEEE Symp. Security Privacy*, 2019, pp. 1415–1430.

[3] M. Antonakakis *et al.*, "Understanding the Mirai botnet," in *Proc. USENIX Security Symp.*, 2018, pp. 1093–1110.

[4] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, "Measurement and analysis of hajime, a peer-to-peer IoT botnet," in *Proc. Netw. Distrib. Syst. Security Symp.*, 2019, pp. 1–9.

[5] S. Oltan, P. Mittal, and H. V. Poor, "BlackIoT: IoT botnet of high wattage devices can disrupt the power grid," in *Proc. USENIX Security Symp.*, 2018, pp. 15–32.

[6] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "SoK: Security evaluation of how-based IoT deployment," in *Proc. IEEE Symp. Security Privacy*, 2019, pp. 1–19.

[7] J. Obermaier and M. Hutle, "Analyzing the security and privacy of cloud-based video surveillance systems," in *Proc. 2nd ACM IoTPTS*, 2016, pp. 22–28.

[8] M. Surbatovich, J. Aljuraidan, L. Bauer, A. Das, and L. Jia, "Some recipes can do more than spoil your appetite: Analyzing the security and privacy risks of IFTTT recipes," in *Proc. 26th Int. World Wide Web (WWW) Conf.*, 2017, pp. 1501–1510.

[9] C. Zuo, W. Wang, Z. Lin, and R. Wang, "Automatic forgery of cryptographically consistent messages to identify security vulnerabilities in mobile services," in *Proc. Netw. Distrib. Syst. Security Symp. (NDSS)*, San Diego, CA, USA, Feb. 2016, p. 4.

[10] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to Android," in *Proc. 17th ACM Conf. Comput. Commun. Security (CCS)*, Chicago, IL, USA, Oct. 2010, pp. 73–84.

[11] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in Android applications," in *Proc. 20th ACM Conf. Comput. Commun. Security (CCS)*, Berlin, Germany, Oct. 2013, pp. 73–84.

[12] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of Google play," in *Proc. ACM SIGMETRICS*, 2014, pp. 221–233.

[13] J. Max, "Backdooring the frontdoor hacking a "perfectly secure" smart lock," in *Proc. DEFCON*, 2016, p. 9.

[14] S. P. Kavalaris and E. Serrelis, "Security issues of contemporary multimedia implementations: The case of sonos and SonosNet," in *Proc. Int. Conf. Inf. Security Digit. Forensics (ISDF)*, 2014, pp. 79–86.

[15] A. Costin, J. Zaddach, A. Francillon, D. Balzarotti, and S. Antipolis, "A large-scale analysis of the security of embedded firmwares," in *Proc. 23rd USENIX Security*, San Diego, CA, USA, Aug. 2014, pp. 95–110.

[16] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt, "On the security of RC4 in TLS and WPA," in *Proc. 22th USENIX Security*, Washington, DC, USA, Aug. 2013, pp. 1–24.

[17] M. Ryan, "Bluetooth smart: The good, the bad, the ugly$\cdots$ and the fix," in *Proc. BlackHat*, 2013.

[18] M. O. Ojo, S. Giordano, G. Procissi, and I. N. Seitanidis, "A review of low-end, middle-end, and high-end IoT devices," *IEEE Access*, vol. 6, pp. 70528–70554, 2018.

[19] A. Milburn, N. Timmers, and N. Wiersma, "There will be glitches: Extracting and analyzing automotive firmware efficiently," in *Proc. BlackHat*, 2018.

[20] J. Shim, K. Lim, J. Jeong, S. Cho, M. Park, and S. Han, "A case study on vulnerability analysis and firmware modification attack for a wearable fitness tracker," *IT Converg. Pract.*, vol. 5, no. 4, pp. 25–33, 2017.

[21] F. Brasser, K. B. Rasmussen, A. Sadeghi, and G. Tsudik, "Remote Attestation for low-end embedded devices: The prover's perspective," in *Proc. 53th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Austin, TX, USA, 2016, pp. 1–6.

[22] Y. Lee, Y. Kim, and J. kim, "Implementation of TLS and DTLS on Zephyr OS for IoT devices," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, 2018, pp. 1292–1294.

[23] K. Ly and Y. Jin, "Security challenges in CPS and IoT: From end-node to the system," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Pittsburgh, PA, USA, 2016, pp. 63–68.

[24] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Understanding Linux malware," in *Proc. IEEE Symp. Security Privacy (SP)*, San Francisco, CA, USA, 2018, pp. 161–175.

[25] *TCG Roots of Trust Specification*, Trusted Comput. Group, Beaverton, OR, USA, Jul. 9, 2018.

[26] S. J. Johnston, M. Scott, and S. J. Cox, "Recommendations for securing Internet of Things sevices using commodity Hardware," in *Proc. IEEE 3rd World Forum Internet Things (WF-IoT)*, Reston, VA, USA, 2016, pp. 307–310.

[27] L. Kbarda, P. Hnyk, L. Vojtech, Z. Lokaj, M. Neruda, and T. Zitta, "Software implementation of a secure firmware update solution in an IoT context," *Inf. Commun. Technol. Services*, vol. 14, no. 4, pp. 369–389, 2016.

[28] *Arm Platform Security Architecture Security Model 1.0 Alpha-2*, Arm Ltd., Cambridge, U.K., 2019.

[29] *Arm Security Technology Building a Secure System Using TrustZone Technology*, Arm Ltd., Cambridge, U.K., 2009.

[30] D. Cerdeira, N. Santos, P. Fonseca, and S. Pinto, "SoK: Understanding the prevailing security vulnerabilities in trustzone-assisted TEE systems," in *Proc. IEEE Symp. Security Privacy (SP)*, San Francisco, CA, USA, 2020, pp. 1416–1432.

[31] *NuMaker-PFM-M2351 User Manual Rev 1.00*, Nuvoton, Inc., Taipei City, Taiwan, 2018.

[32] *TrustZone Technology for ARMv8-M Architecture Version 2.0*, Arm Ltd., Cambridge, U.K., 2017.

[33] *OneM2M—Home*. [Online]. Available: http://www.onem2m.org/

[34] OneM2M Partners, "TS-0002-Requirements-V4.6.0," 2019.

[35] W. Zhang, Y. Chen, H. Li, Z. Li, and L. Sun, "PANDORA: A scalable and efficient scheme to extract version of binaries in IoT firmwares," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Kansas City, MO, USA, 2018, pp. 1–6.

[36] S. Vasile, D. Oswald, and T. Chothia, "Breaking all the things—A systematic survey of firmware extraction techniques for IoT devices," in *Proc. Int. Conf. Smart Card Res. Adv. Appl. (CARDIS)*, Montpellier, France, 2018, pp. 171–185.

[37] D. Strobel, D. Oswald, B. Richter, F. Schellenberg, and C. Paar, "Microcontrollers as (in)security devices for pervasive computing applications," *Proc. IEEE*, vol. 102, no. 8, pp. 1157–1173, Aug. 2014.

[38] N. Sklavos, K. Touliou, and C. Efstathiou, "Exploiting cryptographic architectures over hardware vs. software implementation: Advantages and trade-offs," in *Proc. 5th WSEAS Int. Conf. Appl. Elect. Eng.*, Prague, Czech Republic, 2006, pp. 147–151.

[39] D. A. Wheeler, "Preventing heartbleed," *IEEE Comput.*, vol. 47, no. 8, pp. 80–83, Aug. 2014.

[40] *W75F32W 32M-bit Secure Serial Flash Memory Security Target Revision B*, Winbond Electron. Corporat., Taichung City, Taiwan, 2017.

[41] *ESP8266 Technical Reference Version 1.4*, Espressif Inc., Shanghai, China, 2019.

[42] E. Bertino and N. Islam, "Botnets and Internet of Things security," *Computer*, vol. 50, no. 2, pp. 76–79, Feb. 2017.

[43] *Water Meter Threat Model and Security Analysis (English language Protection Profile) Beta-1*, Arm Ltd., Cambridge, U.K., 2018.

[44] *Arm Platform Security Architecture Trusted Base System Architecture for ARMv6-M, ARMv7-M and ARMv8-M 1.0 Beta-1*, Arm Ltd., Cambridge, U.K., 2019.

[45] C. Kil, E. C. Sezer, A. M. Azab, P. Ning, and X. Zhang, "Remote attestation to dynamic system properties: Towards providing complete system integrity evidence," in *Proc. IEEE/IFIP Int. Conf. Depend. Syst. Netw.*, 2009, pp. 115–124.

[46] *Arm Versatile Express Cortex-M Prototyping Systems (V2M-MPS2 and V2M-MPS2+) Technical Reference Manual*, Arm Ltd., Cambridge, U.K., 2016.

[47] *Application Note AN521 Example CoreLink SSE-200 Subsystem for MPS2+*, Arm Ltd., Cambridge, U.K., 2018.

[48] *ARM MBED-CRYPTO Library*. [Online]. Available: https://github.com/ARMmbed/mbed-crypto

Junyoung Jung received the B.S. degree from the Department of Electronic Engineering, Kyung Hee University, Yongin-si, South Korea, in 2018, where he is currently pursuing the M.S. degree.

His research interests include embedded system security, IoT security, and smart car security.

Beomseok Kim received the B.S., M.S., and Ph.D. degrees from the Department of Computer Engineering, Kyung Hee University, Yongin-si, South Korea, in 2010, 2012, and 2016, respectively.

He was an Assistant Professor with the Department of Computer Software Engineering, Changshin University, Changwon, South Korea, from 2016 to 2018, and a Research Professor with the Department of Computer Engineering, Kyung Hee University in 2019. In 2020, he joined as a Research Professor with the Department of Computer Science Engineering, JeonBuk National Univeristy, Jeonju, South Korea, and currently an Instructor with the Department of Computer Engineering, Kyung Hee University. His research interests include wireless sensor and body area networks, embedded systems, and network security.

Jinsung Cho received the B.S., M.S., and Ph.D. degrees in computer engineering from Seoul National University, Seoul, South Korea, in 1992, 1994, and 2000, respectively.

He was a Visiting Researcher with IBM T. J. Watson Research Center, Ossining, NY, USA, in 1998 and a Research Staff with SAMSUNG Electronics, Suwon-si, South Korea, from 1999 to 2003. He is currently a Professor with the Department of Computer Engineering, Kyung Hee University, Yongin-si, South Korea. His research interests include IoT security, system security, and embedded systems.

Ben Lee (Member, IEEE) received the B.E. degree in electrical engineering from the Department of Electrical Engineering, State University of New York at Stony Brook, Stony Brook, NY, USA, in 1984, and the Ph.D. degree in computer engineering from the Department Electrical and Computer Engineering, Pennsylvania State University, State College, PA, USA, in 1991.

He is a Professor and an Associate School Head of the School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA. His research interests include multimedia streaming, wireless networks, embedded systems, computer architecture, multithreading and thread-level speculation, and parallel and distributed systems.

Prof. Lee was a recipient of the Loyd Carter Award for Outstanding and Inspirational Teaching in 1994, the Alumni Professor Award for Outstanding Contribution to the College and the University from the OSU College of Engineering in 2005, and the HKN Innovation Teaching Award from Eta Kappa Nu, School of Electrical Engineering and Computer Science in 2008. He has been on the Program Committees and the Organizing Committee for numerous international conferences, including from 2005 to 2012 IEEE Workshop on Pervasive Wireless Networking and the IEEE International Conference on Pervasive Computing and Communications (PerCom). He was the Workshop Chair for PerCom 2009. He was a Guest Editor for the Special Issue on Wireless Networks and Pervasive Computing of the *International Journal of Pervasive Computing and Communications*. He was also an Invited Speaker at the 2007 International Conference on Embedded Software and System and a Keynote Speaker at the 2014 ACM International Conference on Ubiquitous Information Management and Communication. He was the TPC Chair and the General Chair for the 2018 and 2020 Annual IEEE Consumer Communications and Networking Conference (CCNC), respectively. He is also an Adjunct Faculty Member from the Korea Advanced Institute of Science and Technology.