xTSeH: A Trusted Platform Module Sharing Scheme Towards Smart IoT-eHealth Devices

Di Lu, *Member, IEEE*, Ruidong Han, Yulong Shen, *Member, IEEE*, Xuewen Dong, *Member, IEEE*, Jianfeng Ma, *Member, IEEE*, Xiaojiang Du, *Fellow, IEEE*, and Mohsen Guizani, *Fellow, IEEE*

Abstract—**IoT based eHealth system brings a revolution to healthcare industry, with which the old healthcare systems can be updated into smarter and more personalized ones. The practitioners can continue monitoring the physical status of the patients at anytime and anywhere, and develop more precise treatment plans by analyzing the collected data, such as heart rate, blood pressure, blood glucose. Actually, these smart sensors used in eHealth system are smart embedded devices (SED). Due to the limitations on hardware capabilities, these inter-connected SEDs lack of security considerations in design and implementation, and face the threats from the network. To prevent the malicious users (or programs) from tampering with the SEDs, trusted platform module (TPM) is adopted, which can guarantee the system integrity via detecting unauthorized modifications to data and system environment. However, due to the limited scalability and insufficient system resources, not all SEDs can be deployed with TPM chips. To address this issue, in this paper, a TPM extension scheme (xTSeH) is proposed. In xTSeH, we have extended the functions of a TPM deployed in a SED (TSED) to those non-TPM-protected SEDs (N-TSED) via network. A shadow TPM in the form of a kernel module is designed as the trust base for the N-TSED, which is the representative of the TPM in TSED. Then, three protocols are proposed to implement the integrity verification and inter-SED authentication. Finally, a Raspberry Pi based prototype system is designed and implemented. The feasibility and usability of our scheme are proved by the analysis of the experimental results of system performance.**

Index Terms—**Internet of Things (IoT), smart eHealth device, trusted platform module, remote attestation, TPM sharing.**

Manuscript received December 15, 2019; revised March 11, 2020; accepted March 14, 2020. Date of publication September 4, 2020; date of current version January 15, 2021. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 61602357, Grant 61972310, Grant 61602364, Grant 61902290, Grant 61902291, and Grant U1736216; in part by the Key Research and Development Program of Shaanxi Province under Grant 2019ZDLGY12-04, Grant 2019ZDLGY12-03, Grant 2019ZDLGY13-01, and Grant 2019ZDLGY13-06; in part by the Open Project from the Key Laboratory of Grain Information Processing and Control (Henan University of Technology), Ministry of Education, under Grant KFJJ-2016-202; in part by the China Postdoctoral Science Foundation Funded Project under Grant 2016M592762; and in part by the Key Program of NSFC-Tongyong Union Foundation under Grant U1636209. *(Corresponding authors: Yulong Shen; Jianfeng Ma.)*

Di Lu is with the School of Computer Science and Technology, Xidian University, Xi'an 710071, China, and also with the Key Laboratory of Grain Information Processing and Control, Ministry of Education, Henan University of Technology, Zhengzhou 450001, China (e-mail: dlu@xidian.edu.cn).

Ruidong Han, Yulong Shen, and Xuewen Dong are with the School of Computer Science and Technology, Xidian University, Xi'an 710071, China (e-mail: hanruidong@stu.xidian.edu.cn; ylshen@mail.xidian.edu.cn; xwdong@xidian.edu.cn).

Jianfeng Ma is with the School of Network Engineering, Xidian University, Xi'an 710071, China (e-mail: jfma@mail.xidian.edu.cn).

Xiaojiang Du is with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122 USA (e-mail: dxj@ieee.org).

Mohsen Guizani is with the Department of Computer Science and Engineering, Qatar University, Qatar (e-mail: mguizani@ieee.org).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2020.3020658

I. Introduction

IOT, as a network of interconnected things, plays a pivotal role in connection between physical world and cyberspace. It allows people and things to be connected with anything, with any network use, from anywhere, and time and anyone to transmit information and data. Everything in the physical world is becoming connected with others globally due to use of internet. All these advantages are guaranteed by the increasingly powerful IoT smart sensors with capabilities of data collection, transmission, processing and storage. With these "smart" features, IoT also brings a new revolution to healthcare industry. For example, the IoT can assist the existing healthcare system by developing flexible remote Patient Monitoring System (PMS) that can benefit the patients by continuous collecting physical data (such as heart rate, blood pressure and body temperature) and getting quick medical responses from the medical practitioners. Figure 1 gives a schematic of an IoT based PMS, in which a smart phone performs as the sink device aggregating the physical data and transmitting the data to the practitioner for the best decision and further treatments.

In spite of many benefits provided by healthcare systems, nevertheless, there are vulnerable to a wide range of security threats due to their portability and design. An IoT based PMS usually faces the threats from three levels, data collection level (DCL), data transmission level (DTL) and data storage level (DSL). The attacks in DCL emerges in data collection step, such as altering information, dropping some important data, resending data message. In DTL, the attackers can spy data by eavesdropping, interrupting communication among devices, sending extra signals to block the base station and networking traffic. However, in DSL, the attackers can intrude the IoT device to modify patient medical data, change the configuration of the device's system etc., which can cause PMS device to malfunction, and the forged data will mislead the doctor to give correct treatments. In this paper, we focus on the integrity of the PMS data in DSL.

To address the data integrity issue, the trusted platform module (TPM) has been widely used in general-purpose

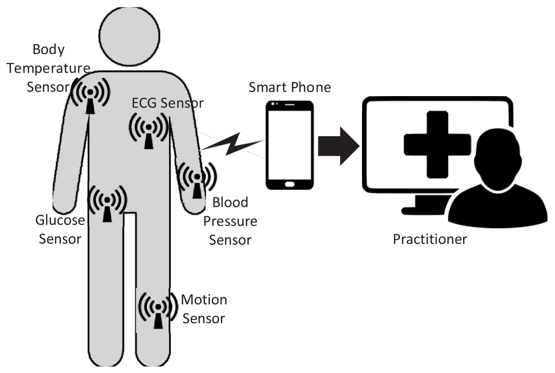


Fig. 1. Schematic of the IoT in Patient Monitoring System.

computing platforms, such as servers and personal computers, to perform as a tamper-proof hardware trust base to ensure the integrity of the host system and provide cryptographic services to upper-level applications. Unlike the dedicated trusted computing technologies, such as Intel SGX, ARM TrustZone, TPM chip can be widely used in a variety of computing systems, not only in general-purpose computers but also in embedded systems, due to the low coupling between the TPM and the host system. Especially for a PMS system consisting of smart embedded devices (SED, the sensors are usually embedded systems), which usually lack of security design and run in an open network environment, the TPM can protect significant data, applications, critical system firmware from being modified by attackers or malwares, as well as unauthorized changes to the hardware environment.

Considering a PMS with IoT smart sensors (which are also SEDs. In the following, to generalize our research work, we use SED instead of IoT smart sensor), to guarantee the data and system integrity, each SED in the network prefers to be protected by a secure module which is solid and tamper-proof so that the malicious changes to the system can be effectively detected. For example, there is a kind of attack towards wireless sensors named *Sensor Worm Attacks (SWA)* [1], by which an attacker can inject malicious code into the sensor system via exploiting the software vulnerabilities that are commonly discovered in popular sensor OS libraries (e.g., TinyOS). The injected malicious code can hijack the program execution and self-propagate to other sensors via network. A trusted module can be used to solve this problem which can detect any modifications to OS and applications. Thus, software-based approaches are proposed to ensure the SED's memory content being untampered, such as SWATT [2] and SCUBA [3]. However, software trusted module cannot ensure the security of itself once the OS has been intruded or tampered. Hence, to adopt a hardware trusted module, which can be considered as a tamper-proof one, is a considerable solution.

However, due to the limited scalability or insufficient system resources, it is hard to deploy hardware TPM to such smart embedded devices. To address this issue, we present a TPM sharing scheme, named xTSeH(**Ex**tending **T**rust to **S**mart **eH**ealth System), in which the functions of the TPM deployed in a SED (TSED) can be extended to the non-TPM-protected

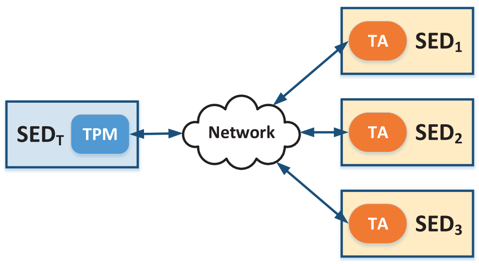


Fig. 2. Extending TPM from TSED to N-TSED via network.

SEDs (N-TSED) to establish a solid trusted environment via network connection. Figure 2 shows that a TPM is extended from the TSED to three N-TSEDs via network. SED_T denotes the TSED equipped with a TPM chip, whereas SED_x denote N-TSEDs, each of which has a *Trusted Agent (TA)* performing as the representative of the TPM in SED_T. Thus, in order to ensure the data security in the procedure of TPM sharing in the untrusted network and to implement a well-protected TA in the N-TSED are the key issues to be solved in our scheme.

The rest of this paper is organized as follows. The threat model, background and our assumptions are presented in section II. The design and implementation of xTSeHare described in section III. After that, we will evaluate the prototype of xTSeHin section IV. In section V, relevant research work will be surveyed and finally our paper is concluded in section VI.

II. Background

A. Threats

In this paper, we suppose that an attacker can invade a SED by exploiting and utilizing hardware/software vulnerabilities and modify system and user data for their attacking aims. Moreover, a SED can be physically captured and installed an extra hardware (e.g., a expanding-board for stealing or forging data) or the attacker can replace the original hardware with malicious ones for their attacking goals, such as forging user data, changing system behaviours by modifying configurations, being controlled by the remote attacker for a long-term attack, etc. These malicious modifications to the system environment will lead to SED malfunctioned and even the failure of the entire SED network.

B. Assumptions

Firstly, we assume that the network composed of SEDs is untrusted and faces the threats from the attackers regardless of whether the network is wired or wireless. Also, the N-TSED is supposed to be untrusted. Secondly, we assume that the goals of the attacks towards the SED network is modifying data, stealing data or changing system behavior by tampering with system configurations (the compromised SED disguises to be a valid node), thus, an attacker is assumed not to replace the operating system with a modified one, disable node functions or damage the hardware of a N-TSED whether the node is physically captured or intruded via network. This assumption is reasonable, because these behaviors will cause significant changes to the system which will be easily detected or make the device unusable which is meaningless to the attackers. Hence, in our threat model, attacks are supposed to modify software or hardware environment. Thirdly, the TSED node is assumed to be trusted in the network and not to be compromised. This assumption is pivotal, since the TSED provides trusted services for the entire network which is the basis of our TPM extending scheme. Fourthly, we assume that system update of N-TSED cannot be carried out in an open network environment (e.g., in the Internet or public WiFi), which may provide opportunities for the potential malicious visitors to intrude into the system and tamper with the updating operations. Instead, in order to guarantee the integrity and authenticity of the system metrics generated based on the system configurations, updating operations (including updating OS and applications as well as updating the hardware, such as replacing malfunctioned components) are only allowed to be executed in an isolated and controlled environment, in which the threats from possible attackers can be eliminated. For example, when updating the system (software/hardware) of the drones in a UAV collaboration network, according to our assumption, the more secure way is that updating operations should be carried out when all the drones have returned the base and are off-line or in a controlled and isolated intranet. Finally, each node is assumed to be online as well as the compromised nodes, since an offline node makes no sense to the attackers.

III. TPM Extending (xTSeH) Scheme

In this section, we first briefly introduce the TPM and then discuss the details of the design and implementation of the TPM extending scheme xTSeH.

A. Trusted Platform Module Basics

The Trusted Platform Module (TPM) is a piece of microchip which is proposed by Trusted Computing Group (TCG) [4] and used to measure the integrity of a computing platform. It is designed to be a micro, secured SoC, which provides cryptographic operations, such as asymmetric key generation, data encryption/decryption, signing and migration of keys between TPMs, as well as random number generation and hashing. It also provides secure storage for a small amount of data, such as cryptographic keys. The objective of a TPM is to provide a hardware-based root-of-trust for a computing platform, which is considered to be more tamper-proof compared with software-based trusted computing base. With a TPM, a trust chain can be established from the hardware level of a system to the software environment, which can be also extended to other computing platforms via *remote attestation*. Thus, all systems of the network can be proved to be *trusted*.

In order to facilitate the use of TPM, the Trust Computing Group (TCG) has proposed a software stack (TCG Software Stack, TSS [4]) architecture for easily calling the functions of TPM rather than using TPM commands which are more complicated. Figure 3 gives the diagram of this architecture. In a typical TCG trusted application, TSS locates between the hardware TPM and applications, which contains three layers:

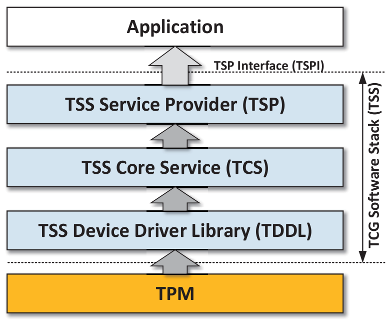


Fig. 3. The architecture of TCG software stack (TSS).

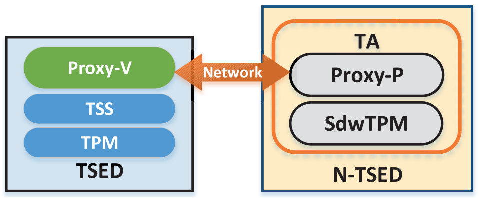


Fig. 4. The architecture of xTSeH.

TSS Device Driver Library (TDDL), TSS Core Service (TCS) and TSS Service Provider (TSP). The TDDL is provided by the corresponding manufacturer and provides standard interfaces between system kernel mode and user mode. With these interfaces, all TPMs (produced by different manufacturers) look and behave the same from the high-level callers. With the interfaces provided by TDDL, a service module named TCS is established, which provides all primitives and other sophisticated functions of TPM as services. In order to provide rich and easy-to-use interfaces for user applications, a TSP is introduced which calls the services provided by TCS. TSP offers C/C++-compliant interfaces (TSPI), so that the user can program with these interfaces to use TPM functions.

In our scheme, we use TSPI functions to implement some cryptographic operations, such as hashing, PCR extending, data encryption/decryption. Since TSPI is a TCG standard interfaces which eliminate the hardware specific differences among TPM chips from various manufacturers, the applications developed with TSPI can be compiled with any version-compliant TSPI libraries and executed on any TCG specification-compliant TPMs. In the discussion of each protocol implementation in section III-C, we will discuss the TSPI functions used in these protocols in detail.

B. System Design

Figure 4 shows the architecture of our TPM sharing scheme. TSED is equipped with a hardware TPM chip and deployed a TSS, which provides APIs for accessing TPM functions. A proxy, named *Proxy-V*, is designed to perform as a verifier which interacts with each TA deployed in the N-TSED to verify the integrity and authenticity of the N-TSED. These verification operations are finished by calling TSPI functions of the TSS.

For N-TSED, a trusted agent is designed to perform as the trust base, which guarantees the integrity of the local system via communicating with Proxy-V. Shown as the figure, TA contains two key components: *Proxy-P* and *SdwTPM*. SdwTPM (refers to *Shadow TPM*) is a kernel module which performs as a representative of the remote hardware TPM and starts automatically with the OS and collects system configuration information (SysCI) for generating system metric of the current N-TSED. An application-level proxy, Proxy-P (refers to prover proxy), is designed to act as a bridge between Proxy-V and SdwTPM.

C. xTSeHImplementation

The key of TPM-sharing relies on how to attest the integrity and authenticity of the N-TSED with TPM in TSED in a secure manner. Hence, in this section, three protocols: *Trusted Booting Protocol (TBP)*, *Remote Verification Protocol (RVP)* and *Node Authentication Protocol (NAP)* are proposed and discussed.

1) Trusted Booting Protocol: Trusted Booting Protocol (TBP) will be executed when the N-TSED starts and finally reports whether the system configurations of N-TSED have been tampered. Algorithm 1 gives the implementation of TBP, and table I shows the essential notations and their descriptions.

As shown in the algorithm, five primary entities are involved: the N-TSED (D_N), the TSED (D_T), Shadow TPM ($SdwTPM$) and two proxies, $ProxyP$ and $ProxyV$. Moreover, several TPM functions are involved in this protocol:

- $\mathsf{TPM\_PcrExtend}(M_i)$: this function computes a digest by hashing M_i (system configuration) into a specific Platform Configuration Register (PCR) of TPM. This operation is also known as *PCR Extend*, and can be described as the following equation:

$$digest_{new} = \mathsf{Hash}_{\mathsf{hashAlg}}(digest_{old} \| data_{new}).$$

  The function $\mathsf{Hash}_{\mathsf{hashAlg}}$ can be different hash algorithms, such as SHA1, SHA256. With this function, the final digest will be stored in the selected PCR.
- $\mathsf{NV\_Read}(Index_{DN})$: this function loads a pre-stored data with index $Index_{DN}$ from the built-in non-volatile storage of TPM.
- $\mathsf{Data\_Unseal}(V_{NV})$: there is an important prerequisite before invoking this function, that the data must be encrypted by $\mathsf{Data\_Seal}()$. Therefore, we assume that the data V_{NV} read by $\mathsf{NV\_Read}()$ has been encrypted before. Thus, function $\mathsf{Data\_Unseal}()$ can decrypt V_{NV} by offering the corresponding key.

Firstly, D_N loads the OS kernel and $SdwTPM$ module. After that, $ProxyP$ is launched. Then, $SdwTPM$ module checks local system environments by invoking $\mathsf{CheckSysEnv}()$, including whether the process $ProxyP$ is running and whether the device is online. Note that, these are essential prerequisites for executing protocol TBP. As mentioned in figure 4, $ProxyP$ is the core process which has to handle all the datagram between D_N and D_T, and cannot be terminated. Hence, if $ProxyP$ fails to run or is shut down unexpectedly, the system of D_N may have been compromised and tampered. Moreover, we require that D_N must be online, so that $ProxyV$ can verify it via $ProxyP$.

Algorithm 1 Trusted Booting Protocol

```
Require: $D_N$, $D_T$, $SdwTPM$, $ProxyP$, $ProxyV$
Ensure: $D_N$ is successfully verified compromised or not.
1: • $D_N$ executes:
2: $D_N$ starts booting procedure, including loading $SdwTPM$ module etc.
3: • $SdwTPM$ executes:
4: if $\mathsf{CheckSysEnv}()$ fails then
5:     Prompting warnings and shutdown the device
6: end if
7: $M_N$ ← system configuration data, and $M_N = \{M_1, M_2, \ldots, M_k\}$
8: sending $M_N$ to $ProxyP$
9: • $ProxyP$ executes:
10: Computing $R_{D_N} = \{ID_{D_N}, \mathcal{N}_{D_N}, TS_{D_N}, E_{D_N}, Sig_{D_N}\}$,
    in which $E_{D_N} = \mathsf{Enc}(AIK^{Pub}_{D_T}, M_N)$ and
    $Sig_{D_N} = \mathsf{Enc}(AIK^{Pri}_{D_N}, \mathsf{Hash}(ID_{D_N} \| \mathcal{N}_{D_N} \| TS_{D_N} \| E_{D_N}))$
11: Sending $R_{D_N}$ to $ProxyV$
12: • $ProxyV$ executes:
13: Checking the freshness and integrity of $R_{D_N}$
14: $M^*_N$ ← $\mathsf{Dec}(AIK^{Pub}_{D_N}, E_{D_N})$
15: for $i = 1$ to $k$ do
16:     $V_{PCR}$ ← $\mathsf{TPM\_PcrExtend}(M_i)$
17: end for
18: $V_{NV}$ ← $\mathsf{NV\_Read}(Index_{D_N})$
19: $V_{D_N}$ ← $\mathsf{Data\_Unseal}(V_{NV})$
20: if $V_{D_N} \neq V_{PCR}$ then
21:     $msg_{RSP}$ = false
22: else
23:     $msg_{RSP}$ = true
24: end if
25: $ProxyV$ sends $RSP_{D_N}$ to $D_N$, that
    $RSP_{D_N} = \{ID_{D_N}, msg_{RSP}, f(\mathcal{N}_{D_N}), Sig_{D_T}\}$, in which
    $Sig_{D_T} = \mathsf{Enc}(AIK^{Pri}_{D_T}, \mathsf{Hash}(ID_{D_N} \| msg_{RSP} \| f(\mathcal{N}_{D_N})))$
```

TABLE I
A List of Essential Notations Used in TBP

Notation	Description
D_N, D_T	N-TSED and TSED
$SdwTPM$	The shadow TPM
$Proxy\text{–}P$, $Proxy\text{–}V$	The proxies residing in D_N and D_T
M_N,M_i	The SysCI vector and its element i
$AIK^{Pub}_{D_N}$,$AIK^{Pri}_{D_N}$	The attestation identity key pair of D_N
ID_{D_N}	The identity of D_N
$\mathcal{N}_{D_N}$	A nonce generated by D_N
TS_{D_N}	The timestamp generated by D_N
Sig_{D_N},Sig_{D_T}	The signatures created by D_N and D_T
$Index_{D_N}$	The index of TPM NV-storage preserving D_N's SysCI
msg_{RSP}	The response value (**true** of **false**)

Hence, an offline D_N will be treated as an abnormal device and should be shut down due to security considerations. In summary, once $\mathsf{CheckSysEnv}()$ fails, D_N will be terminated by $SdwTPM$, otherwise, $SdwTPM$ will collect and compute SysCI data, including key hardware serial numbers, operating system version, the hash of the booting partition, the hash of $ProxyP$ (both of program file and runtime process), etc, which will be wrapped into a vector M_N. Each element of M_N, M_i $(1 \leqslant i \leqslant k)$, denotes a type of system configuration. Finally, $SdwTPM$ generates system report message R_{D_N} and sends it to $ProxyP$. Note that, in R_{D_N}, parameters: $\mathcal{N}_{D_N}$, TS_{D_N}, E_{D_N} and Sig_{D_N} are used to guarantee the freshness, confidentiality, integrity and authenticity of the message, which will be discussed in *Security Discussion* of the TBP. Once $ProxyV$ has received R_{D_N}, it will verify the freshness and integrity of the message as following:

1) $ProxyV$ decrypts Sig_{D_N} and obtains the hash value denoted as H_{D_N}
2) $ProxyV$ computes $H^*_{D_N}$ =Hash$(ID_{D_N} \| \mathcal{N}_{D_N} \| TS_{D_N} \| E_{D_N})$
3) if $H_{D_N} \neq H^*_{D_N}$, the integrity verification fails
4) if $\mathcal{N}_{D_N}$ has been received before or timestamp TS_{D_N} has expired, $ProxyV$ will discard this message and request $ProxyP$ to send the message again. Moreover, if the verification fails for l times (l is a configurable threshold), D_N will be considered as a compromised node.

After that, $ProxyV$ decrypts E_{D_N} and obtains the SysCI data of D_N in plain-text. Then, $ProxyV$ continuously calls TPM function TPM_PcrExtend() to extend each SysCI data M_i to the selected PCR to generate the system metrics of D_N, which is denoted as V_{PCR}. Then, $ProxyV$ reads the encrypted pre-stored configuration data of D_N (denoted as V_{NV}) from the non-volatile storage of the TPM chip and decrypts V_{NV} to obtain V_{D_N} with the TPM's SRK (Storage Root Key) by calling Data_Unseal(). Next, $ProxyV$ compares V_{PCR} with V_{D_N} to determine whether D_N has been tampered or not, and then generates the response message RSP_{D_N} which will be finally sent to D_N. Note that, in RSP_{D_N}, function $f(x)$ performs certain calculation on x, such as simply increasing x by 1. Thus, the receiver, D_N, can determine whether this message is refresh. The signature Sig_{D_T} will be used to check the integrity and authenticity of the message by D_N.

Security Discussion: In this part, we mainly discuss the possible attacks to the TBP and how the protocol defends against these attacks as follows:

1) Attack to $ProxyP$: Two major approaches are used to attack $ProxyP$, tampering with and terminating the proxy. For the first case, the SysCI data that the $SdwTPM$ collects contains a hashed value generated from $ProxyP$ program file, which will change when the proxy is tampered and can be detected by $ProxyV$ during its verification procedure (line 15 to 24 in algorithm 1). For the latter case, the termination of the proxy can be detected by the $SdwTPM$ in function CheckSysEnv().
2) Attack to R_{D_N}: There are two major ways to attack message R_{D_N}, including modifying values within R_{D_N} and forging R_{D_N}. For the first case, E_{D_N} cannot be modified since the attacker cannot decrypt E_{D_N}. Sig_{D_N} still can't be tampered by the attacker due to lacking of $AIK^{Pri}_{D_N}$. Besides, any modifications to any other values (such as ID_{D_N}, $\mathcal{N}_{D_N}$ and TS_{D_N}) will be easily detected by $ProxyV$ while verifying the hash value within Sig_{D_N}. Supposing an attacker forges R_{D_N}, the signature Sig_{D_N} can't be forged due to lacking of $AIK^{Pri}_{D_N}$, even if other values are easy to be fabricated.
3) A Disguised TSED (a fake D_T): Assuming an attacker pretends to be a legitimate TSED, since it doesn't have $AIK^{Pri}_{D_T}$, it cannot decrypt E_{D_N} in message R_{D_N} and it cannot generate a signature of legitimate TSED in the response message (RSP_{D_N}) either.
4) Attack to SdwTPM module: as introduced above, the shadow TPM (SdwTPM) is a kernel module which performs as a representative of the hardware TPM of the TSED (according to our assumption, the TSED cannot be compromised). However, assuming an attacker successfully intruding into a N-TSED by exploiting the system vulnerabilities and obtaining the root privilege, the SdwTPM is also in the risk of being tampered by the attacker. However, according to protocol TBP, the SdwTPM has to collect and compute SysCI data which also includes the digest of the SdwTPM itself, and send the SysCI data to the hardware TPM of the TSED. Thus, once the SdwTPM is tampered, the corresponding digest will be changed which will not be matched with that stored in the non-volatile storage of the hardware TPM. In this case, the malicious modifications to the SdwTPM will be detected when the TSED performs verifications on the SysCI of N-TSED and the compromised SdwTPM. Thus, the compromised device will be added to a block list, which will be not allowed to access to the TSED again. Besides, other nodes can learn from the TSED that which device has been compromised and will not communicate with it in the future. Hence, the security mechanism of our scheme will not be damaged even if there is a compromised device and the SdwTPM within. From this perspective, the integrity of SdwTPM is still protected by the remote hardware TPM within the TSED.

TABLE II
A LIST OF ESSENTIAL NOTATIONS USED IN RVP (THE SAME OR SIMILAR NOTATIONS AS TABLE I ARE OMITTED)

Notation	Description
m_{REQ}	Request message created by D_T
M_N^{REQ},M_i^{REQ}	The requested SysCI vector and its element i

2) Remote Verification Protocol: Remote Verification Protocol (RVP) will be executed by $ProxyV$ when one device needs to verify the integrity of the target device. For example, a TSED, as the unique hardware trust root, periodically verifies other N-TSEDs to ensure that no compromised device exists. Once such a node is found, it will be marked as an compromised node and added into the blacklist by the TSED. Algorithm 2 shows the implementation of the RVP, and table II gives the essential notations and their descriptions (the same or similar notations as in table I are omitted).

Compared with algorithm 1, at the beginning of algorithm 2, $ProxyV$ first generates a message REQ_{D_N} for D_N to request

Algorithm 2 Remote Verification Protocol

```
Require: $D_N$, $D_T$, $SdwTPM$, $ProxyP$, $ProxyV$
Ensure: $D_N$ is successfully verified by $D_T$, compromised or not.
1:  • $ProxyV$ executes:
2:  Generating verification request message $REQ_{D_N} = \{m_{REQ}, ID_{D_N}, \mathcal{N}_{REQ}, TS_{REQ}, Sig_{REQ}\}$, in which $Sig_{REQ} = \mathsf{Enc}(AIK_{D_T}^{Pri}, \mathsf{Hash}(m_{REQ} \| ID_{D_N} \| \mathcal{N}_{REQ} \| TS_{REQ}))$
3:  Sending $REQ_{D_N}$ to $D_N$
4:  • $ProxyP$ executes:
5:  Verifying the integrity and authenticity of $REQ_{D_N}$ via $Sig_{REQ}$
6:  if verification fails then
7:      $REQ_{D_N}$ is compromised, discarding the message
8:  end if
9:  Extracting requested SysCI set $M_N^{REQ}$ from $m_{REQ}$, and $M_N^{REQ} = \{M_i^{REQ} | i = 1, 2, \ldots, k\}$
10: Sending $M_N^{REQ}$ to $SdwTPM$
11: • $SdwTPM$ executes:
12: $M_N \leftarrow$ collecting SysCI according to $M_N^{REQ}$
13: Sending $M_N$ to $ProxyP$
14: • $ProxyP$ executes:
15: Generating response message $RSP_{D_N}$, $RSP_{D_N} = \{ID_{D_N}, f(\mathcal{N}_{REQ}), TS_{D_N}, E_{M_N}, Sig_{D_N}\}$, in which $E_{M_N} = \mathsf{Enc}(AIK_{D_T}^{Pub}, M_N)$ and $Sig_{D_N} = \mathsf{Enc}(AIK_{D_N}^{Pri}, ID_{D_N} \| f(\mathcal{N}_{REQ}) \| TS_{D_N} \| E_{M_N})$
16: Sending $RSP_{D_N}$ to $ProxyV$
17: • $ProxyV$ executes:
18: Checking the freshness and integrity of response message $RSP_{D_N}$
19: $M_N^* \leftarrow \mathsf{Dec}(AIK_{D_T}^{Pri}, E_{M_N})$
20: for $i = 1$ to $k$ do
21:     $V_{PCR} \leftarrow \mathsf{TPM\_PcrExtend}(M_i^*), (M_i^* \in M_N^*)$
22: end for
23: $V_{NV} \leftarrow \mathsf{NV\_Read}(Index_{D_N})$
24: $V_{D_N} \leftarrow \mathsf{Data\_Unseal}(V_{NV})$
25: if $V_{D_N} \neq V_{PCR}$ then
26:     marking $D_N$ as a compromised device
27: else
28:     successful verification
29: end if
```

the SysCI of D_N. Note that the requested SysCI is contained in m_{REQ} (line 2), which is determined by $ProxyV$ according to its needs. For example, if $ProxyV$ needs to verify the integrity of $ProxyP$, the $SdwTPM$ module, the boot partition of the TF-card and the CPU serial number of $ProxyP$ are selected as the SysCI to be verified, thus, m_{REQ} can be

$$m_{REQ} = \{H_{ProxyP}, H_{SdwTPM}, H_{boot}, I_{SN-CPU}\},$$

in which H_x denotes the hash value of x (e.g., H_{SdwTPM} signifies the hash value of the $SdwTPM$ module), whereas I_y denotes a string value (mostly in plain-text) of y (e.g., I_{SN-CPU} signifies the serial number of CPU). In REQ_{D_N}, Sig_{REQ} will be verified by $ProxyP$ of the N-TSED to confirm the authenticity and integrity of the message. Once REQ_{D_N} is proven legitimate, $ProxyP$ notifies $SdwTPM$ to collect the requested SysCI into M_N according to m_{REQ}. After that, $ProxyP$ generates the response message RSP_{D_N}, in which the SysCI set, M_N, is included. Besides, random number $\mathcal{N}_{REQ}$ will be processed in order to guarantee the freshness of the message for the further verification by $ProxyV$. Also, E_{M_N} ensures the confidentiality of the submitted SysCI, whereas Sig_{D_N} indicates that the message is from a real D_N rather than a fake one. Then, $ProxyV$ begins to verify the received SysCI, M_N^* (line 20-29), which is the same as that in algorithm 1 (line 15-24). If the verification fails, $ProxyV$ will mark the N-TSED (D_N) as a compromised one, which will be added to a blacklist and cannot be accessed by other nodes anymore.

Security Discussion: In this part, we present the discussion on several typical attacks to this protocol, and how to defend against them. Some attacks, such as attack to $ProxyP$ and disguised TSED, have been discussed in TBP.

1) Attack to REQ_{D_N}: Firstly, considering an attacker tries to tamper with request message REQ_{D_N} by modifying m_{REQ}, receiver's ID (ID_{D_N}), nonce $\mathcal{N}_{REQ}$ and time stamp TS_{REQ} (these parameters are given in plain-text), the modifications can be detected when $ProxyP$ recomputes the hash value of this message, which will be compared with the hash value within the signature Sig_{REQ}. Moreover, any modifications to nonce $\mathcal{N}_{REQ}$ will also be detected by $ProxyV$ while verifying $f(\mathcal{N}_{REQ})$. Secondly, Sig_{REQ} cannot be forged, since the attacker doesn't have the private key of D_T. Finally, if the attacker replays the message to $ProxyP$, the receiver will find that the $\mathcal{N}_{REQ}$ and TS_{REQ} have been received before.
2) Attack to RSP_{D_N}: Firstly, similar with our discussion above, any unauthorized modifications to the parameters of the message, such as ID_{D_N}, $f(\mathcal{N}_{REQ})$, will change the hash value derived from these parameters, and will be easily detected by $ProxyV$ while verifying the hash value within the signature Sig_{D_N}. Then, E_{M_N} cannot be decrypted by the attacker due to lacking of $AIK_{D_T}^{Pri}$. Similarly, an attacker also cannot fake the signature of the response message due to lacking of $AIK_{D_N}^{Pri}$.

3) Node Authentication Protocol: Supposing a N-TSED D_A intends to access another N-TSED D_B, D_A has to authenticates the identity of D_B via the TSED, which preserves the SysCI of all the other nodes and performs as the authentication center of the network. To realize the authentication between any two N-TSEDs, *Node Authentication Protocol (NAP)* is proposed in algorithm 3. Table III gives the essential notations and their descriptions (the same or similar notations as in table I are omitted).

In algorithm 3, three main parts are involved in the protocol, the challenger D_A, the prover D_B and the verifier D_T. D_A requests D_T to authenticate D_B via verifying the SysCI of D_B. If the authentication is successful, a session key K_{AB}

Algorithm 3 Node Authentication Protocol

```
Require: $D_A$, $D_B$, $D_T$, $ProxyP_A$, $ProxyP_B$, $ProxyV$
Ensure: $D_B$ is successfully authenticated by $D_A$ via $D_T$, compromised or not.
1: • $ProxyP_A$ executes:
2: Generating request message $REQ^{IDA}_{D_A} = \{ID_{D_A}, ID_{D_B}, \mathcal{N}_{D_A}, TS_{D_A}, Sig_{D_A}\}$, in which $Sig_{D_A}$=Enc($AIK^{Pri}_{D_A}$,Hash($ID_{D_A}\|ID_{D_B}\|\mathcal{N}_{D_A}\|TS_{D_A}$))
3: Sending $REQ^{IDA}_{D_A}$ to $ProxyV$
4: • $ProxyV$ executes:
5: $b_{ret} \leftarrow$ verifying $Sig_{D_A}$ in $REQ^{IDA}_{D_A}$
6: ErrHandler($b_{ret}$)
7: $b_{ret} \leftarrow$RVP($D_A$,$M^{REQ}_A$)
8: ErrHandler($b_{ret}$)
9: $b_{ret} \leftarrow$ checking whether $D_B$ is in the blacklist
10: ErrHandler($b_{ret}$)
11: $b_{ret} \leftarrow$RVP($D_B$,$M^{REQ}_B$)
12: ErrHandler($b_{ret}$)
13: Creating $RSP_{D_A}$ =CreateRspMsg($E_{D_A}$,$D_A$,$D_B$)
14: Creating $RSP_{D_B}$ =CreateRspMsg($E_{D_B}$,$D_A$,$D_B$)
15: Sending $RSP_{D_A}$ and $RSP_{D_B}$ to $D_A$ and $D_B$ respectively
16: • $ProxyP_A$ executes:
17: $b_{ret} \leftarrow$ verifying $Sig_{D_T}$ in $RSP_{D_A}$
18: if $b_{ret}$ = false then
19:     $RSP_{D_A}$ has been tampered. Restarting this protocol.
20: end if
21: $b_{ret}$ =Dec($AIK^{Pri}_{D_A}$,$E_{D_A}$)
22: if $b_{ret}$ = false then
23:     Obtaining the session key $K_{AB}$
24: else
25:     Authentication fails
26: end if
27: • $ProxyP_B$ executes:
28: $b_{ret} \leftarrow$ verifying $Sig_{D_T}$ in $RSP_{D_B}$
29: if $b_{ret}$ = false then
30:     $RSP_{D_B}$ has been tampered. Discarding the message.
31: end if
32: $K_{AB} \leftarrow$ Dec($AIK^{Pri}_{D_B}$,$E_{D_B}$)
```

TABLE III
A LIST OF ESSENTIAL SYMBOLS USED IN NAP (THE SAME OR SIMILAR SYMBOLS AS TABLE I ARE OMITTED)

Symbol	Description
D_A,D_B	The N-TSED A and B
b_{ret}	The return value of the algorithm (boolean)
K_{AB}	The session key between D_A and D_B

will be created for the further communication between D_A and D_B, otherwise, D_A will be informed that D_B is a compromised device.

In D_A's request $REQ^{IDA}_{D_A}$, ID_{D_B} is included in order to indicate the device it intends to authenticate. Once the request message is received, $ProxyV$ first checks if the message is from a real D_A by decrypting Sig_{D_A} and verifies the integrity of the message via the hash value in Sig_{D_A}. If the verification fails, $ProxyV$ will call ErrHandler(b_{ret}), in which a response message RSP_{D_A} for D_A is created by CreateRspMsg(m,x) and an encrypted error message $E^{Err}_{D_A}$ =Enc($AIK^{Pub}_{D_A}$,Msg_{Err}). Functions CreateRspMsg(m,x) and CreateRspMsg(m,x,y) generate a response message with message content m and the receiver x, whereas y denotes the device related to D_A. For example, in line 13, the parameter "y" is D_B which is the device being authenticated by D_A. Then, $ProxyV$ checks the platform integrity of D_A via RVP protocol. After that, $ProxyV$ checks whether D_B is in the blacklist and measures the platform of D_B via the RVP protocol. After successful measurement, $ProxyV$ generates two response messages RSP_{D_A} and RSP_{D_B} for D_A and D_B respectively which include encrypted session key. Equation 1 shows the contents of both RSP_{D_A} and RSP_{D_B},

$$\begin{cases} RSP_{D_A} = \{ID_{D_A}, ID_{D_B}, f(\mathcal{N}_{D_A}), TS_{D_T}, E_{D_A}, Sig_{D_T}\} \\ RSP_{D_B} = \{ID_{D_B}, ID_{D_A}, f(\mathcal{N}_{D_B}), TS_{D_T}, E_{D_B}, Sig_{D_T}\} \end{cases} \tag{1}$$

in which E_{D_A} and E_{D_B} are defined as equation 2. Note that m_{RSP} contains information about successful authentication on both D_A and D_B, and K_{AB} is a session key (a symmetric key) for the future communication between D_A and D_B.

$$\begin{cases} E_{D_A} = \text{Enc}(AIK^{Pub}_{D_A}, m_{RSP}\|K_{AB}) \\ E_{D_B} = \text{Enc}(AIK^{Pub}_{D_B}, m_{RSP}\|K_{AB}) \end{cases} \tag{2}$$

Signature Sig_{D_T} will be used to verify the authenticity of the sender ($ProxyV$) by D_A and D_B. Then, $ProxyV$ sends both of RSP_{D_A} and RSP_{D_B} to D_A and D_B respectively. Once D_A has received RSP_{D_A}, the signature within the message has to be verified, so does D_B.

Security Discussion: To defense against the attacks to $ProxyP$ and a disguised TSED can refer to that in security discussion of TBP. Similarly, to defend against the attacks to $REQ^{IDA}_{D_A}$, RSP_{D_A} and RSP_{D_B} can refer to that in security discussion in RVP.

IV. EVALUATION

A. Experiment Setup

We have implemented the prototype system, including a kernel module (SdwTPM), $ProxyP$ and $ProxyV$, in Raspbian operating system with kernel 4.19, which is deployed in three RaspBerry Pi 3b+ boards. An Infineon SLB9645 extending board is adopted as the hardware TPM, which is installed in the TSED, and the shadow TPM is implemented as a kernel module which is deployed in the N-TSED and will be loaded automatically by the kernel. Moreover, Trousers 0.3.14 is used as the TCG Software Stack (TSS) to implement the proxies, $ProxyP$ and $ProxyV$. All the components are implemented in C, which have over 2500 lines of source code in total.

Figure 5 shows the deployment of the prototype system (including software and devices) for our evaluations. Three RaspBerry Pis are interconnected via a wireless access point, in which two Pis perform as N-TSED having been installed with shadow TPM (SdwTPM), and one Pi acts as the TSED

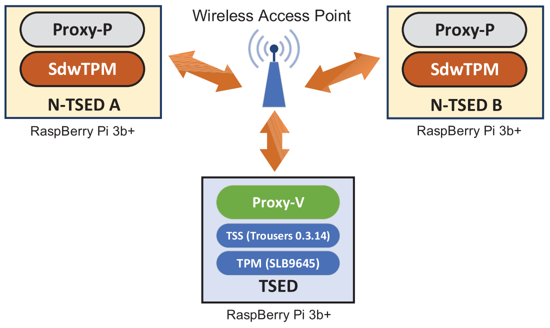


Fig. 5. The deployment of the prototype system for evaluations.

equipped with a hardware TPM (Infineon SLB9645) extending board. Our experiments concentrate on the performance evaluation of protocol execution, including the time consumption of executing the key stages of each protocol as well as that of an entire protocol execution.

B. Result Analysis

In this section, the analysis on our experimental results is presented. For each protocol, we give the performance evaluations on time consumption for each key stage, and then provide the total time cost of each protocol. Finally, the experimental results are discussed.

1) Trusted Booting Protocol: figure 6 shows time cost variations of each key stage in TBP (referring to algorithm 1), including checking environment (ChkEnv, line 4-6), building R_{DN} (Bld-RDN, line 9-10), decrypting & verifying R_{DN} (Dec&Ver-RDN, line 13-14), checking system metrics by TPM (TPMChk, line 15-24) and data transmission (DTrans, line 11). As shown in figure 6(a), it can be learned from the line in chart that the time consumption of stage ChkEnv is nearly 0.2ms, which can be almost ignored. Moreover, It can be found that there are three distinct "steps" on the line at the position near 400, 600 and 800 on the x-axis, which is caused by other running processes of the Raspbian OS, since the operations of $\mathsf{CheckSysEnv}()$ take very short time and can be easily influenced by other system processes having more workload. For figure 6(b), we can observe that the data line fluctuates near 21ms, which implies that the overall trend of time consumption on encryption operations within stage Bld-RDN is stable, even if other system processes may impact it. Furthermore, in figure 6(c), the time cost of stage Dec&Ver-RDN fluctuates near 13.5ms. Then, figure 6(d) shows the performance of verifying system metric of D_N by TPM chip, which fluctuates near 600ms. Since in-TPM micro-controller unit (MCU), PCR and NV-storage modules run at a lower speed than the on-board MCU (4 cores, 1.4GHz), it takes more time for stage TPMChk to finish the verification. Figure 6(e) gives the performance of network transmission between two nodes, $ProxyP$ and $ProxyV$. Interestingly, it is obvious that the data line of stage DTrans is almost flat except some glitches, and the time consumption is about 22ms. Due to our stable network condition (the devices in our experiment are fixed and the wireless channel is dedicated and not occupied by other devices), there is no large fluctuation in the data transmission rate, and the glitches are caused by network communications of other running tasks of the operating system, which brings trivial impacts to our evaluations. Finally, figure 6(f) gives the total time consumption of executing TBP, and we can see that the data line fluctuates near about 650ms, which approximately equals to the sum of time consumption of all stages in TBP.

TABLE IV
THE WEIGHTED AVERAGE TIME CONSUMPTION OF EACH STAGE IN TBP

Stage	Weighted Avg. (ms)
ChkEnv	0.2
Bld-RDN	20.99
Dec&Ver-RDN	13.64
TPMChk	596.76
DTrans	22.16
Total	645.63

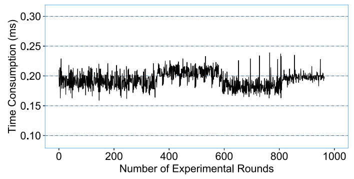
Figure 7 gives corresponding histograms of the statistics of time consumption for each stage. With these figures, we use equation 3 to compute the weighted average value of time consumption for each stage and the entire process of each protocol.

$$T = \sum_{i=1}^{n} \bar{v}_i \times \frac{t_i}{P} \tag{3}$$

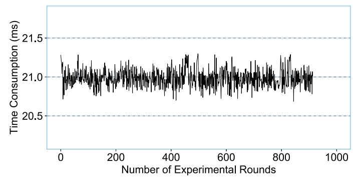
$\bar{v}_i$ denotes the arithmetic mean of the i^{th} time range, e.g., in figure 7(b), $\bar{v}_5 = \frac{6.14+6.18}{2} \approx 6.16$. t_i denotes the value of times corresponding to $\bar{v}_i$, e.g., $t_5 = 188$, and P is the total number of protocol executions, e.g., 1000 rounds of execution per protocol.

With equation 3, we obtain the weighted average time for each stage as well as that of the entire protocol shown as table IV. The first column indicates different stages, whereas the second gives the weighted average time consumption. It is obvious that stage TPMChk takes the most of the time (about 92.4%) to run TBP. Hence, in-TPM operations are the bottleneck of the protocol execution. Moreover, TPM is a serial device which can be occupied by one task at a time, thus, the protocol performance will be influenced by this shortage.

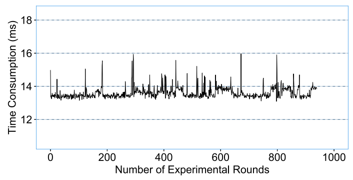
Furthermore, intuitively, figure 7(a), 7(b) and 7(c) show obvious characteristics of normal distribution, which reveal that the results are consistent with central limit theorem, that various separate factors in the experimental environment can make results look normal. We can find that the histograms in figure 7(a) and 7(b) are symmetric, since both of the stages are based on software implementation, which use on-board MCU and can be influenced by the other OS processes using the same MCU. Since these influences are independent of our program, the histograms show symmetric characteristics. However, for figure 7(c), the histogram is asymmetric (log normal distribution), that the peak (the topmost bar) is to on the left side of the x-axis (less time cost). In fact, this is caused by multi-thread implementations of stage Dec&Ver-RDN and TPMChk. In $ProxyV$, once a request from $ProxyP$ has arrived, a thread will be created to process the request by $ProxyV$.



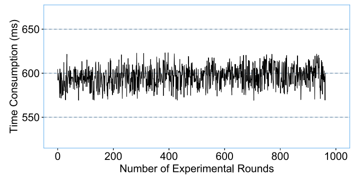
(a) The changes of time consumption on stage ChkEnv in TBP.



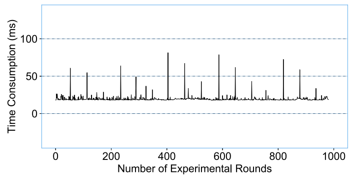
(b) The changes of time consumption on stage Bld-RDN in TBP.



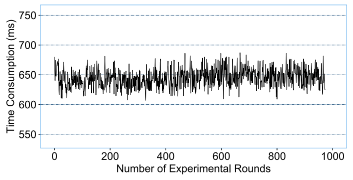
(c) The changes of time consumption on stage Dec&Ver-RDN in TBP.



(d) The changes of time consumption on stage TPMChk in TBP.

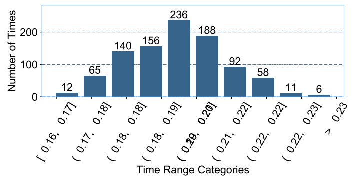


(e) The changes of time consumption on stage DTrans in TBP.

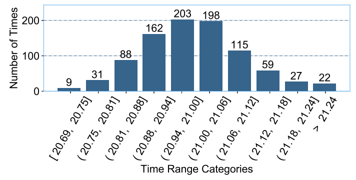


(f) The changes of total time consumption on executing TBP.

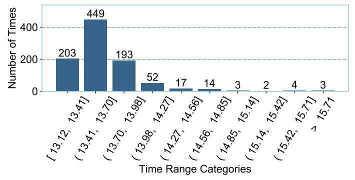
Fig. 6. The changes of the time consumption on five stages in TBP as well as the total.



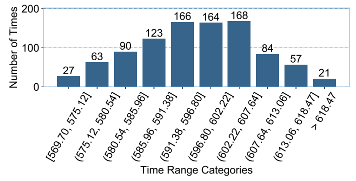
(a) The histogram of time consumption on stage ChkEnv in TBP.



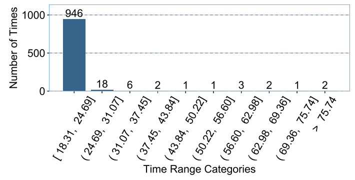
(b) The histogram of time consumption on stage Bld-RDN in TBP.



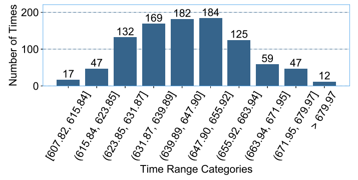
(c) The histogram of time consumption on stage Dec&Ver-RDN in TBP.



(d) The histogram of time consumption on stage TPMChk in TBP.



(e) The histogram of time consumption on stage DTrans in TBP.



(f) The histogram of total time consumption on executing TBP.

Fig. 7. The histograms of the time consumption on five stages in TBP as well as the total.

Figure 8 gives a time diagram of multi-thread based implementations of stage Dec&Ver-RDN and TPMChk. In the diagram, four threads (Thx) start sequentially at the same interval T_{Int}, and $T_{D\&V}$ denotes the time cost of executing Dec&Ver-RDN, whereas T_{TPMChk} denotes that of system metric verification in TPM. Moreover, T_w indicates the waiting time for the thread to start using TPM, which is an increasing variable, since the TPM chip can only be used serially. With the multi-thread implementation, we accelerate the procedure of Dec&Ver-RDN in a multi-device scenario, thus, in figure 7(c), most of the experimental results are at the left position, in range [13.12-13.7ms]. However, since TPM chip can be only accessed serially, the time cost of stage TPMChk of each thread cannot be accelerated. In that case, stage TPMChk becomes the bottleneck of our implementation and consumes most of the time in executing TBP. Besides, because the operations in stage TPMChk are all performed in TPM chip, which are only influenced by the TPM capabilities (independent factors), the histogram in figure 7(d) is symmetric. Let's see the statistics about stage DTrans in figure 7(e). The histogram is a log normal distribution, in which 94.6%

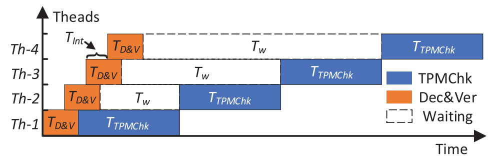
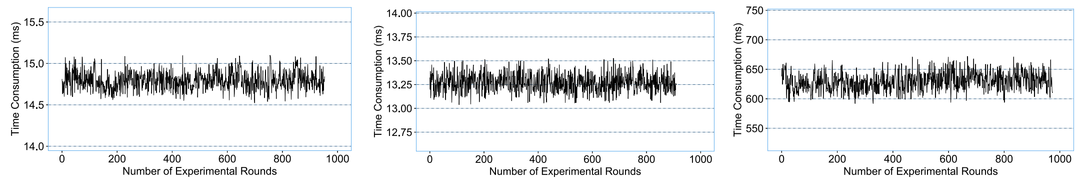


Fig. 8. Multi-thread execution of stage Dec&Ver-RDN and TPMChk.

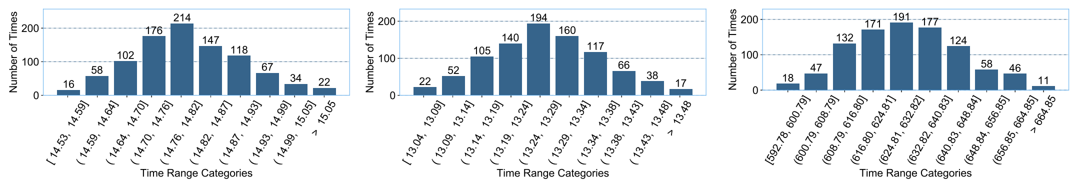
experiment results are in the first range [18.31-24.69ms]. The reason for this phenomenon is that the network resource is only occupied by our program, and the other OS processes seldom use network. Therefore, exclusive network occupation causes network transmission to be fast and stable, which also makes the histogram asymmetric. Finally, the total time consumption statistics is given in figure 7(f), which is also a symmetric histogram. This is because the most time consumption is mainly in stage TPMChk, which is a symmetric histogram and determines the distribution of the total time cost.

2) Remote Verification Protocol: Similar to TBP, we divide RVP (shown as algorithm 2) into six stages, 1) $ProxyV$ generating REQ_{D_N} (Bld-ReqDN, in line 1-2), 2) $ProxyP$



(a) The changes of time consumption on the stage of Bld-ReqDN in RVP. (b) The changes of time consumption on the stage of Dec&Ver-ReqDN in RVP. (c) The changes of total time consumption of executing RVP.

Fig. 9. The changes of the time consumption on two key stages in RVP as well as the total.



(a) The histogram of time consumption on the stage of Bld-ReqDN in RVP. (b) The histogram of time consumption on the stage of Dec&Ver-ReqDN in RVP. (c) The histogram of total time consumption of executing RVP.

Fig. 10. The histograms of the time consumption on two key stages in RVP as well as the total.

decrypting & verifying REQ_{DN} (Dec&Ver-ReqDN, in line 4-9), 3) $SdwTPM$ collecting SysCI (SdwTPM-SysCI, in line 11-12), 4) $ProxyP$ building response message (Bld-RspDN, line 14-15), 5) $ProxyV$ decrypting & verifying response message (Dec&Ver-RspDN, line 17-19) and 6) the TPM within $ProxyV$ checking SysCI of $ProxyP$ (TPMChk$_{\mathrm{PxyP}}$, in line 20- 29). Actually, in RVP, stage Bld-RspDN, Dec&Ver-RspDN and TPMChk$_{\mathrm{PxyP}}$ have similar procedure as the stage Bld-RDN, Dec&Ver-RDN and TPMChk in TBP, thus, the performance evaluations about these stages can refer to the corresponding discussions (about Bld-RDN, Dec&Ver-RDN and TPMChk) in section IV-B.1. Besides, since stage SdwTPM-SysCI, which is implemented in the kernel module SdwTPM, has the similar operations as the stage ChkEnv does, its performance can refer to stage ChkEnv in table IV. In the following, the implementation details of stages ChkEnv and SdwTPM-SysCI are presented.

- ChkEnv: directory /proc is parsed to find whether process $ProxyP$ is in running state; file /usr/class/net/wlan0/operstate is adopted to determine whether the network connection is in "up" state ("wlan0" signifies the wireless network interface name).
- SdwTPM-SysCI: to collect system metrics, files /proc/version, /proc/cpuinfo and /proc/meminfo are used to obtain information about OS version, MCU and memory, which will be sent to $ProxyP$.

According to the implementations above, we can find that both of the stages ChkEnv and SdwTPM-SysCI are based on parsing files, thus, they have similar performance.

Figure 9 gives the line charts that describe the changes of the time consumption on stage Bld-ReqDN and Dec&Ver-ReqDN as well as the total time cost of RVP. It is obvious that the time consumption of running Bld-ReqDN (figure 9(a)) and Dec&Ver-ReqDN (figure 9(b)) fluctuates in ranges 14.5-15.0ms and 13.0-13.5ms respectively, and there is

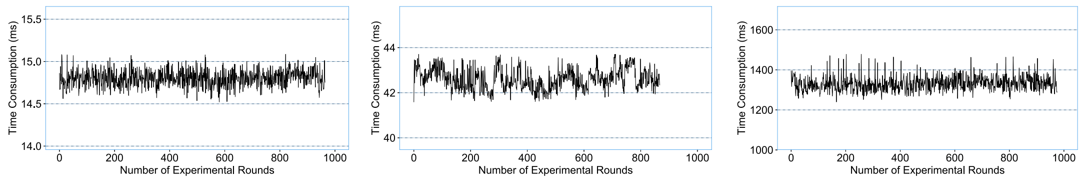
TABLE V
THE WEIGHTED AVERAGE TIME CONSUMPTION OF EACH STAGE IN RVP

Stage	Weighted Avg. (ms)
Bld-ReqDN	14.80
Dec&Ver-ReqDN	13.28
SdwTPM-SysCI*	refers to ChkEnv
Bld-RspDN*	refers to Bld-RDN
Dec&Ver-RspDN*	refers to Dec&Ver-RDN
TPMChk$_{\mathrm{PxyP}}$*	refers to TPMChk
Total	630.33

little difference between the two ranges (about 1.5ms). This is because both of the stages have similar operations inside (encryption for Bld-ReqDN and decryption for Dec&Ver-ReqDN). Also, the stages are easily impacted by the other OS running processes. Moreover, the sending procedure (e.g., line 3 in algorithm 2) between two nodes can refer to our discussion on figure 6(e) and 7(e). Figure 9(c) shows the total time consumption of executing RVP, which fluctuates around 625ms and approximately equals to that of TBP (figure 6(b)).

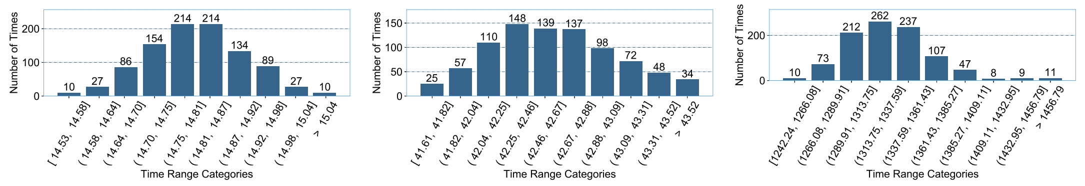
Figure 10 gives the histograms associated with the line charts in figure 9. Also, these histograms are nearly symmetric, because the OS processes that influence the stage execution can be treated as an independent factors. Then, we use equation 3 to compute the weighted average of time cost for stages Bld-ReqDN and Dec&Ver-ReqDN, and the whole RVP execution.

Table V gives the weighted average time cost for each stage of RVP. Note that the stages marked "*" have similar performance as that of TBP listed in Weighted Avg. Besides, we can see that both Bld-ReqDN and Dec&Ver-ReqDN have similar performance (14.80ms and 13.28ms), and the total time cost of RVP is 630.33ms, which is also approximately equals to that of TBP (645.63ms). The tiny difference on performance



(a) The changes of time consumption on the stage of Bld-ReqDA in NAP. (b) The changes of time consumption on the stage of SKDist in NAP. (c) The changes of total time consumption on executing NAP.

Fig. 11. The changes of time consumption on two key stages in NAP as well as the total.



(a) The histogram of time consumption on the stage of Bld-ReqDA in NAP. (b) The histogram of time consumption on the stage of SKDist in NAP. (c) The histogram of total time consumption on executing NAP.

Fig. 12. The histograms of the time consumption on two key stages in NAP as well as the total.

between the two protocols is caused by the extra stage ChkEnv of TBP, which takes a bit more time (about 15ms).

3) Node Authentication Protocol: In the experiments on NAP, we divide algorithm 3 into four stages, 1) $ProxyPA$ building request message $REQ_{D_A}^{IDA}$ (Bld-ReqDA, in line 1-2), 2) $ProxyV$ verifying $REQ_{D_A}^{IDA}$, authenticating D_B and distributing session key K_{AB} to D_A and D_B via RSP_{D_A} and RSP_{D_B} (SKDist, in line 4-15), 3) $ProxyP_A$ verifying RSP_{D_A} and obtaining K_{AB} (Ver&ObtSK-DA, in line 16-26) and 4) $ProxyP_B$ verifying RSP_{D_B} and obtaining K_{AB} (Ver&ObtSK-DB, in line 27-32). Similarly, from the implementation of algorithm 1 and 3, we can find that stages Ver&ObtSK-DA and Ver&ObtSK-DB have almost the same procedure as stage Dec&Ver-RDN, thus, the performance of the two stages can refer to the discussion on stage Dec&Ver-RDN of TBP in section IV-B.1. Therefore, in the following, we mainly discuss the experimental results on stages Bld-ReqDA and SKDist.

Figure 11 gives line charts about the changes of time consumption on stages Bld-ReqDA and SKDist of NAP as well as the entire protocol process. In figure 11(a), the data line fluctuates in range 14.5-15.0ms, which is a very small range of variation. Hence, due to such a tiny range, the performance of executing Bld-ReqDA is relatively stable. Then, figure 11(b) depicts the time variation in evaluating stage SKDist. In this stage (line 4-15 in algorithm 3), RVP is adopted (in line 7 and 11) to authenticate nodes D_A and D_B, which are not included in the evaluation of stage SKDist (the performance of RVP can refer to IV-B.2). Hence, the evaluations for this stage only contain $ProxyV$ decrypting and verifying $REQ_{D_A}^{IDA}$ and creating response message for D_A and D_B, which include the cryptographic operations listed as follows.

- Decryption: verifying signature Sig_{D_A} in $REQ_{D_A}^{IDA}$ in line 5 of algorithm 3.
- Encryption: in CreateRspMsg(), creating Sig_{D_T}, E_{D_A} and E_{D_B}.
- Key Generation: generating the session key K_{AB}.

TABLE VI
THE WEIGHTED AVERAGE TIME CONSUMPTION OF EACH STAGE IN NAP

Stage	Weighted Avg. (ms)
Bld-ReqDA	14.81
SKDist (except RVP)	42.63
Ver&ObtSK-DA*	refers to Dec&Ver-RDN
Ver&ObtSK-DB*	refers to Dec&Ver-RDN
Total	1334.01

Hence, these cryptographic operations make the time cost higher than the previous stage Bld-ReqDA, which has only one encryption operation. Figure 11(c) shows that the data line fluctuates near 1300ms, which is relatively stable even if there are some strong glitches caused by local system environment, such as other running processes, network communication interference.

Figure 12(c) gives histograms related to the line charts in figure 11(c). Obviously, these histograms are symmetric due to the same reason that we have discussed in analyzing TBP and RVP experimental results. Similarly, with equation 3, we can obtain the weighted average time cost for each stage of NAP listed in table VI. Unsurprisingly, the time cost of stage Bld-ReqDA is 14.81ms, which approximately equals to that of stage Bld-ReqDN in RVP, since they have nearly identical implementations. However, for stage SKDist, the average time cost is about 42.63ms, which is almost three times as high as that of stage Bld-ReqDN, since there are more cryptographic operations in SKDist. Finally, the total time consumption of NAP is about 1334.01ms, in which there are two calls to RVP for authenticating D_A and D_B. According to the performance of RVP in table V, the time consumption of executing RVP twice is about 1260ms, which occupies the most of the total time cost of NAP.

V. Related Work

A. Data Integrity Protection in eHealth System

Monda *et al.* [5] proposes a robust data integrity module to identify correction of errors within a eHealth system.

It checks the rationality of data, replaces manual operations, identifies data quality problems within the eHealth system. Mantas *et al.* [6] develop a data integrity mechanism for eHealth monitoring system in a smart home environment, which applies the agent technology to achieve data integrity by using cryptographic smart cards. Since terminal medical sensors are usually resource-constrained devices, a lightweight integrity authentication scheme [7] for an eHealth device is proposed. This scheme allows device to authenticate each other in order to secure the collection of health-related data, which uses nonces and Keyed-Hash message authentication to check the integrity of authentication exchanges. As for eHealth device, Cao *et al.* [8] assume that cloud servers and medical institutions may tamper with device information to hide the medical malpractice, so they produce a secure cloud-assisted eHealth system based on blockchain-based currencies to ensure the device operational information not to be modified. Besides, [9]–[12] are also blockchain-based approaches, which are designed to address the issue of data integrity. However, due to the limited storage resources of IoT devices, to maintain and process the blockchain data will occupy non-trivial storage space and computing resources with the growth of the blockchain. In addition, to prevent the security issues in multiple eHealth device nodes, Garkoti *et al.* [13] propose a digital watermarking model to ensure the device data in eHealth system not to be modified. Although watermarking based method can ensure any modifications can be recorded in a distributed manner. However, if there is a fake device in network, it will also disseminate false information.

These aforementioned approaches can be considered as software based solutions, which cannot guarantee the integrity of the protector itself without the support of hardware based root-of-trust once an attacker intrudes the device and obtains superuser privileges.

B. Additional Software Based Approaches for Other Types of System

A. Seshadri *et al.* [2] propose a software-based attestation technique (SWATT) to verify the memory contents of embedded devices. SWATT provides memory content attestation without requiring secure hardware and does not need physical access to the device's memory. Another scheme named SCUBA [3] is then proposed, which is a protocol used to determine whether a sensor node is compromised via verifying the correctness of the self-checksum response and the response delays. However, both of SWATT and SCUBA are software based solutions which would be unavailable once the device system has been tampered. On the other hand, the two schemes are very difficult to design and implement correctly in practice [14]. Ahmed *et al.* [15] propose ModChecker for cloud environment, which checks in-memory kernel modules' code integrity in real time without maintaining a database of hashes. It is able to detect any changes in a kernel module's headers and executable content with no impact on the guest operating systems' performance. SecVisor [16]is a small hypervisor to protect the integrity of kernel code, which provides a trusted environment by sandboxing with traditional OS. Sun *et al.* [17] consider to isolate the security context of a container by establishing a Linux namespace, which could maintain independent security policies for each container to govern their security. This scheme is implemented based on Linux Integrity Measurement Architecture (IMA). Farris *et al.* [18] have conducted a thorough investigation of security models based on SDN and NFV. NFV/SDN based solutions towards IoT security threats are also compared with traditional security models. As mentioned in this paper, to prevent the data from being tampered or protect the system not to be injected by malicious code, a secure data/network tunnel is established. These are all software-based approaches, which are still in danger of being tampered once the OS has been intruded and compromised.

Similarly, they are also software-based methods which lack of protection of tamper-proof hardware root-of-trust.

C. TPM/TEE Based Approaches for Other Types of System

Dewan *et al.* [19] present a holistic approach against sophisticated malware, which creates a lightweight hypervisor for fine-grained software runtime memory protection without modifying OS. The proposed data locker component in the hypervisor prevents the sensitive data of program in persistent storage from leaking to rootkits or other malware. This scheme depends on Integrity Measurement Module (IMM) [20] which relies on hardware TPM. Tan *et al.* [14] propose and implement a remote attestation protocol (TRAP), which detects unauthorized modifications in application code running on sensor nodes with the help of the Trusted Platform Module (TPM) in each node. Lin and Wang [21] use TPM to encrypt BIOS password, which will be then stored in TPM NV-storage instead of in BIOS storage for security considerations. Thus, the encrypted password will be still available even if the BIOS battery has been removed. SeTPM [22] is a secure element based TPM implemented by utilizing Java Card technology. It provides integrity measurement for mobile platforms such as Android. It provides a solution to run Trusted Computing based security protocols while supplying a similar security level as provided by a hardware TPM. Benedictis and Lioy [23] present DIVE, an architecture to support integrity verification of Docker containers, which uses well-known trusted computing techniques, such as Linux IMA and TPM. With this solution, the services in the containers can be attested as if they would be running in a physical platform, and their integrity can be well understood by a third-party. Shankar *et al.* [24] focus on the security threats toward VNF, and the related best practical solutions are given. Particularly for the integrity issue of system booting in NFV, TPM based solution is suggested. However, according to this paper, the TPM can be only used to ensure the integrity measurement in system booting or restarting stages, which cannot guarantee integrity when system are in run-time status. These solutions above rely on a local physical TPM, which is not realistic for the resource-constrained embedded systems.

Abera *et al.* [25] propose a novel approach named "DIAT" for ensuring the correctness and integrity of data and the processing of data in the collaborative embedded systems. The primary contributions of this paper are design and implementation of integrity verification on data flow and control flow, which are dynamic in the networked system. To achieve this goal, a minimal TCB is introduced to provide a solid trust of root for the system. However, in the prototype of DIAT, neither the MCU (Microcontroller Unit) nor the peripherals provide hardware-oriented TCB. The paper suggests that TyTAN [26] (a hardware-assist TCB) or ARM TrustZone-M can be used in DIAT as the ideal TCB.

Zhao and Mannan [27] present Inuksuk to protect the run-time data integrity towards computing systems via trusted computing technologies, such as Intel TEE and TPM. In this scheme, The main idea of this paper concentrates on how to armor the data instead of detecting/recovering the data loss, since the vulnerabilities cannot be completely eliminated when the system complexity increases. In Inuksuk, the trusted execution environment (TEE, in the prototype, Intel TXT/AMD SVM are adopted), TPM and self-encrypting drive (SED) are combined in a seamless way and provide a significant leap in the arms-race against malware. However, the TXT/SVM technologies severely rely on the specific CPU (Intel/AMD series), which cannot be used on low-end embedded systems.

Wang *et al.* [28] propose a novel security scheme based on Intel TEE technology (SGX) for protecting the integrity and confidentiality of data within SQLite database. In SGX, CPU can be a tamper-proof root-of-trust of the system, and the enclaved SQLite engine can execute cryptographic operations in more secure manner, nevertheless, this scheme is platform dependent (relies on Intel CPUs) and not suitable for most of IoT devices whose MCUs are usually ARM-based ones.

VI. Conclusion

To address the issue that smart eHealth devices lack of integrity protection towards system and user data, a TPM extending scheme xTSeHis proposed in this paper to establish trusted environment for each device to realize data integrity protection. In this scheme, TPM functions are extended from the device equipped with a hardware TPM (TSED) to those ones lack of TPM protection (N-TSED) due to the limitations on hardware capabilities, to achieve the expansion of trust root (hardware TPM) from a TSED to all the devices across the network. In xTSeH, we design and implement a shadow TPM in form of kernel module for ensuring the integrity of N-TSED, which is a representative of the TPM in TSED. Then, three key protocols (TBP, RVP and NAP) are proposed, which implement the integrity verification and authentication among devices. We have implemented the prototype system with Raspberry Pis, evaluated the time cost of the protocols, and discussed the results. The experimental results of the three protocols are ideal and acceptable, which prove the feasibility and availability of our scheme in practical applications. Besides, although TPM chip can be the bottleneck when the amount of requests increases, we can solve it by setting a buffer to temporarily store requests or by scheduling requests based on their priorities.

References

[1] A. Francillon and C. Castelluccia, “Code injection attacks on Harvard-architecture devices,” in *Proc. 15th ACM Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2008, pp. 15–26.

[2] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, “SWATT: Software-based attestation for embedded devices,” in *Proc. IEEE Symp. Secur. Privacy*, 2004, pp. 272–282.

[3] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla, “SCUBA: Secure code update by attestation in sensor networks,” in *Proc. 5th ACM Workshop Wireless Secur. (WiSe)*, 2006, pp. 85–94.

[4] *TCG Specification Architecture Overview*, T. C. Group, TCG Specification Revision 1, 2007, pp. 1–24.

[5] J. Monda, J. Keipeer, and M. C. Were, “Data integrity module for data quality assurance within an e-Health system in Sub-Saharan Africa,” *Telemedicine e-Health*, vol. 18, no. 1, pp. 5–10, Jan. 2012.

[6] G. Mantas, D. Lymberopoulos, and N. Komninos, “Integrity mechanism for eHealth tele-monitoring system in smart home environment,” in *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, Sep. 2009, pp. 3509–3512.

[7] H. Khemissa and D. Tandjaoui, “A lightweight authentication scheme for E-Health applications in the context of Internet of Things,” in *Proc. 9th Int. Conf. Next Gener. Mobile Appl., Services Technol.*, Sep. 2015, pp. 90–95.

[8] S. Cao, G. Zhang, P. Liu, X. Zhang, and F. Neri, “Cloud-assisted secure eHealth systems for tamper-proofing EHR via blockchain,” *Inf. Sci.*, vol. 485, pp. 427–440, Jun. 2019.

[9] M. Kubendiran, S. Singh, and A. K. Sangaiah, “Enhanced security framework for e-health systems using blockchain,” *J. Inf. Process. Syst.*, vol. 15, no. 2, pp. 239–250, 2019.

[10] U. Javaid, M. N. Aman, and B. Sikdar, “BlockPro: Blockchain based data provenance and integrity for secure IoT environments,” in *Proc. 1st Workshop Blockchain-enabled Networked Sensor Syst. (BlockSys)*, 2018, pp. 13–18.

[11] D. Minoli and B. Occhiogrosso, “Blockchain mechanisms for IoT security,” *Internet Things*, vols. 1–2, pp. 1–13, Sep. 2018.

[12] T. Rana, A. Shankar, M. K. Sultan, R. Patan, and B. Balusamy, “An intelligent approach for UAV and drone privacy security using blockchain methodology,” in *Proc. 9th Int. Conf. Cloud Comput., Data Sci. Eng. (Confluence)*, Jan. 2019, pp. 162–167.

[13] G. Garkoti, S. K. Peddoju, and R. Balasubramanian, “Detection of insider attacks in cloud based e-healthcare environment,” in *Proc. Int. Conf. Inf. Technol.*, Dec. 2014, pp. 195–200.

[14] H. Tan, W. Hu, and S. Jha, “A TPM-enabled remote attestation protocol (TRAP) in wireless sensor networks,” in *Proc. 6th ACM workshop Perform. Monitor. Meas. Heterogeneous Wireless Wired Netw. (PM2HW2N)*, 2011, pp. 9–16.

[15] I. Ahmed, A. Zoranic, S. Javaid, and G. G. R. Iii, “ModChecker: Kernel module integrity checking in the cloud environment,” in *Proc. 41st Int. Conf. Parallel Process. Workshops*, Sep. 2012, pp. 306–313.

[16] A. Seshadri, M. Luk, N. Qu, and A. Perrig, “SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes,” in *Proc. 21st ACM SIGOPS Symp. Oper. Syst. Princ. (SOSP)*, 2007, pp. 335–350.

[17] Y. Sun, D. Safford, M. Zohar, D. Pendarakis, Z. Gu, and T. Jaeger, “Security namespace: Making linux security frameworks available to containers,” in *Proc. 27th USENIX Secur. Symp. (USENIX Secur.)*, 2018, pp. 1423–1439.

[18] I. Farris, T. Taleb, Y. Khettab, and J. Song, “A survey on emerging SDN and NFV security mechanisms for IoT systems,” *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 812–837, 1st Quart., 2019.

[19] P. Dewan, D. Durham, H. Khosravi, M. Long, and G. Nagabhushan, “A hypervisor-based system for protecting software runtime memory and persistent storage,” in *Proc. Spring Simulation Multiconf. Soc. Comput. Int.*, 2008, pp. 828–835.

[20] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, “Design and implementation of a TCG-based integrity measurement architecture,” in *Proc. USENIX Secur. Symp.*, vol. 13, 2004, pp. 223–238.

[21] K.-J. Lin and C.-Y. Wang, “Using TPM to improve boot security at BIOS layer,” in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2012, pp. 376–377.

[22] S. Proskurin, M. Weiß, and G. Sigl, “seTPM: Towards flexible trusted computing on mobile devices based on globalplatform secure elements,” in *Proc. Int. Conf. Smart Card Res. Adv. Appl.* Cham, Switzerland: Springer, 2015, pp. 57–74.

[23] M. De Benedictis and A. Lioy, “Integrity verification of docker containers for a lightweight cloud environment,” *Future Gener. Comput. Syst.*, vol. 97, pp. 236–246, Aug. 2019.

[24] S. Lal, T. Taleb, and A. Dutta, “NFV: Security threats and best practices,” *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 211–217, Aug. 2017.
[25] T. Abera, R. Bahmani, F. Brasser, A. Ibrahim, A.-R. Sadeghi, and M. Schunter, “DIAT: Data integrity attestation for resilient collaboration of autonomous systems,” in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019.
[26] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, “TyTAN: Tiny trust anchor for tiny devices,” in *Proc. 52nd Annu. Design Autom. Conf. (DAC)*, 2015, pp. 1–6.
[27] L. Zhao and M. Mannan, “TEE-aided write protection against privileged data tampering,” 2019, *arXiv:1905.10723*. [Online]. Available: http://arxiv.org/abs/1905.10723
[28] Y. Wang, Y. Shen, C. Su, J. Ma, L. Liu, and X. Dong, “CryptSQLite: SQLite with high data security,” *IEEE Trans. Comput.*, vol. 69, no. 5, pp. 666–678, May 2020.

Di Lu (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science and technology from Xidian University, China, in 2006, 2009, and 2014, respectively. He is currently an Associate Professor with the School of Computer Science and Technology, Xidian University. His research interests include trusted computing, cloud computing, and system and network security.

Ruidong Han received the B.Eng. degree in computer science from Northwest A&F University, China, in 2018. He is currently pursuing the M.S. degree with the School of Computer Science, Xidian University. His research interests include IoT security, trusted computing, and intelligent system security.

Yulong Shen (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science from Xidian University, Xi’an, China, in 2002, 2005, and 2008, respectively. He is currently a Full Professor with the School of Computer Science and Technology, Xidian University, and a Researcher with the State Key Laboratory of Integrated Services Networks. His research interest includes wireless network security. He is a member of ACM. He has also served on the technical program committees of several international conferences, including ICEBE, INCoS, CIS, and SOWN.

Xuewen Dong (Member, IEEE) received the B.E., M.S., and Ph.D. degrees in computer science and technology from Xidian University, China, in 2003, 2006, and 2011, respectively. From 2016 to 2017, he was with Oklahoma State University, USA, as a Visiting Scholar. He is currently an Associate Professor with the School of Computer Science, Xidian University. His research interests include cognitive radio networks, blockchain, wireless network security, and big data privacy.

Jianfeng Ma (Member, IEEE) received the B.S. degree in mathematics from Shaanxi Normal University, China, in 1985, and the M.E. and Ph.D. degrees in computer software and communications engineering from Xidian University, China, in 1988 and 1995, respectively. From 1999 to 2001, he was with Nanyang Technological University, Singapore, as a Research Fellow. He is currently a Professor with the School of Computer Science, Xidian University, China. His current research interests include distributed systems, computer networks, and information and network security.

Xiaojiang Du (Fellow, IEEE) received the B.S. and M.S. degrees in electrical engineering from Tsinghua University, Beijing, China, in 1996 and 1998, respectively, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland at College Park, in 2002 and 2003, respectively. He is currently a tenured Professor with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA. His research interests include security, wireless networks, and systems. He has authored more than 300 journal and conference papers in these areas and a book published by Springer. He has been awarded more than six million U.S. dollars research grants from the U.S. National Science Foundation (NSF), Army Research Office, Air Force Research Laboratory, NASA, Qatar, State of Pennsylvania, and Amazon. He is a Life Member of ACM. He received the Best Paper Award at the IEEE GLOBECOM 2014 and the Best Poster Runner-Up Award at ACM MobiHoc 2014. He serves on the editorial boards of three international journals.

Mohsen Guizani (Fellow, IEEE) received the B.S. (Hons.) and M.S. degrees in electrical engineering and the M.S. and Ph.D. degrees in computer engineering from Syracuse University, Syracuse, NY, USA, in 1984, 1986, 1987, and 1990, respectively. He served in different academic and administrative positions at the University of Idaho, Western Michigan University, the University of West Florida, the University of Missouri-Kansas City, the University of Colorado at Boulder, and Syracuse University. He is currently a Professor with the CSE Department, Qatar University, Qatar. His research interests include wireless communications and mobile computing, computer networks, mobile cloud computing, security, and smart grid. He has authored nine books and more than 500 publications in refereed journals and conferences. He is a Senior Member of ACM. He also served as a member, the chair, and the general chair of a number of international conferences. Throughout his career, he received three teaching awards and four research awards. He also received the 2017 IEEE Communications Society WTC Recognition Award and the 2018 AdHoc Technical Committee Recognition Award for his contribution to outstanding research in wireless communications and ad hoc sensor networks. He was the Chair of the IEEE Communications Society Wireless Technical Committee and of the TAOS Technical Committee. He served as the IEEE Computer Society Distinguished Speaker. He is currently a Distinguished Lecturer of the IEEE ComSoc. He has guest edited a number of special issues in the IEEE journals and magazines. He is currently the Editor-in-Chief of the *IEEE Network Magazine*, serves on the editorial boards of several international technical journals, and the Founder and Editor-in-Chief of the *Wireless Communications and Mobile Computing* (Wiley).