Design and Implementation of Trusted Sensing Framework for IoT Environment

Sungjin Park, Jaemin Park, and Jisoo Oh

Abstract: **Even though Internet of things (IoT) sensing services are introduced in a wide range of areas, it is not applicable to mission-critical services due to the lack of the trustworthiness of the IoT sensing data. To address this problem, we propose TruSense, a novel trusted sensing framework for the IoT environment that covers end-to-end implementation from an IoT device to a cloud service. The TruSense framework includes a small sensing board, a communication protocol, and a cloud service for the trusted sensing in the IoT environment. To show our framework's feasibility, we design an ARM TrustZone-based IoT sensing board and implement an application and a trusted sensing service running in the Secure world. We also implement a cloud service for the trusted sensing in Google app engine, which is one of the widely used cloud services.**

Index Terms: **ARM TrustZone, cloud computing, trusted sensing.**

I. INTRODUCTION

As the Internet of things (IoT) environment matures, the IoT is being introduced in a wide range of areas such as environmental monitoring, logistics (e.g., truck fleet tracking and item location), home automation, and national infrastructures (e.g., power plants and SCADA networks). Generally, IoT devices collect and aggregate data from sensors placed in various environments. The data collected by the IoT devices are sent to a cloud service, and the cloud service creates valuable and meaningful information with the collected data using the big data analytics.

However, in the context of the mission-critical IoT environment, sensing data manipulated by attackers can lead to big disasters. For example, when an attacker can transfer crafted sensing data to a monitoring system, an organization can make wrong decisions. As a result, the monitoring system operates actuators to control critical operations such as gas pipelines and turbines, and it can make critical infrastructures dangerous. As with the infrastructure case, insecure sensors embedded in a medical device can also threaten human life. If a sensor that monitors a patient's status is compromised, a doctor can prescribe the wrong medicines or fail to treat the patient's diseases.

The trusted sensing can be a proactive way to detect and protect such urgent cases. This is a secure channel to guarantee the trustworthiness between a sensor in a device and a cloud service. Many researchers have presented the related studies in the various parts of the trusted sensing: A sensor-side implementation [1], a device-side implementation (e.g., a hypervisor with a virtual machine [2] and a hypervisor with a trusted OS [3]), a server-side implementation [4], a protocol proposal [5], and a cloud service for the mobile device [6]. In addition, ETSI released the first globally applicable standard for consumer IoT [7]. This document recommends that IoT devices should safeguard security-sensitive data using secure, trusted storage mechanisms provided by the trusted execution environment (TEE), the universal integrated circuit card, and so on. All of the aforementioned work addressed various problems such as the integrity of the platform and sensing data, the large trusted computing base (TCB), and the mutually trusted communication channel. Still, there is no practical end-to-end framework that integrates and covers entire parts for the trusted sensing.

In this paper, we design and create an ARM TrustZone-based sensing board, the TruSense device, for the trusted sensing. The TrustZone technology compartments the TCB for sensor-handling in the Secure world from the untrusted operating system (OS) in the Normal world. Sensor-handling code in the TCB captures a sensing value from a sensor dedicated to the Secure world and computes an evidence value with a TCB measurement value, the sensing value, and a device-specific private key. To compute the TCB's integrity value, we establish a secure chain from an application processor (AP) to a secure OS in the Secure world. Specifically, a boot loader in the sensing board measures the secure OS and passes the measurement to the secure OS. The TCB measurement value is stored in secure memory only accessible by the Secure world. When computing the evidence value, the sensor-handling code signs the TCB measurement value and the sensing value with the device-specific private key.

An IoT cloud service needs to collect and aggregate sensing values to generate value-added information via big data analysis or deep learning. To aggregate sensing values from many sensing boards, we implement the aggregator running in Google app engine (GAE) [8]. A sensing board sends a sensing value and an evidence value to the aggregator and the aggregator performs the remote attestation that verifies the evidence value with the corresponding public key. With this verification, the aggregator can assure that the integrity of the sensing board platform and the sensing value is valid. After successful verification, the aggregator inserts the sensing value into a cloud database. For the user convenience, the aggregator provides a user interface via the web browser to show sensing values stored in the database.

The sensing boards communicate with the aggregator via the REST API, which is commonly used for Web services. Moreover, the sensing boards transmit sensing data and remote attestation messages over the HTTPS protocol to secure the communication channel.

Manuscript received June 2, 2020; revised October 19, 2020; approved for publication by Junbeom Hur, Division III Editor, November 23, 2020.

S. Park, J. Park, and J. Oh are with the Affiliated Institute of of Electronics and Telecommunications Research Institute (ETRI), email: {taiji, jmpark, jsoh77}@nsr.re.kr.

J. Park is the corresponding author.

Digital Object Identifier: 10.23919/JCN.2020.000035

The contributions of this paper are as follows.

- **Design of end-to-end trusted sensing framework**: We design a trusted sensing framework to build up the trust of sensing data. Our design covers the Normal world and Secure world in a sensing board, a cloud service, a user interface for the cloud service, and an attestation protocol between the sensing board and the cloud service.
- **Implementation of a TrustZone-enabled sensing board for IoT environment**: We design and implement a sensing board based on the ARM TrustZone technology. This board mainly features a WiFi module, a TrustZone LED indicator, a battery connector, and so on that are required for the IoT environment.
- **End-to-end implementation of the trusted sensing framework for the IoT environment**: We implement a prototype from hardware (i.e., a small-size trusted sensing board) to software (i.e., a cloud service and a software stack running on a sensing board).
- **Practicality of the proposed framework**: We show that our prototype framework runs on GAE, one of the commonly used commercial cloud platforms.

This paper is structured as follows. Section II summarizes related work, Section III explains the ARM TrustZone technology, and Section IV defines our problems. Section V, Section VI, and Section VII describe our approaches to address the problems. Section VIII explains how to implement the end-to-end implementation of the TruSense framework and Section IX evaluates the TruSense framework. In Section X, we conclude this paper and present future work.

II. RELATED WORK

The trusted sensing is a secure channel to guarantee the trustworthiness of measured data between a sensor in a device and a sensing service [3]. By adopting the trusted sensing, an IoT service can gain the benefits of the trusted sensing: *Authentication of the source of the sensing value* and *the integrity of the sensing value from a sensor to a cloud service*. To assure the features above, many researchers conducted related work.

Park *et al.* proposed TGVisor [6], a tiny hypervisor-based trusted sensing framework for the cloud environment, to guarantee the trustworthiness of sensing values. Specifically, TGVisor introduced the dynamic root of trust measurement (DRTM) and the trusted platform module (TPM) to attest the integrity of the TCB (i.e., the hypervisor) and the sensing values. Oh *et al.* [9] proposed a trusted authentication as a service in the cloud on top of a variant of TGVisor. This approach establishes a trusted path between a keyboard in a device and an authentication service in the cloud.

Liu *et al.* [2] presented two abstraction layers, sensor attestation and sensor seal. Sensor attestation guarantees the sensor's integrity and authenticity. Sensor seal encrypts a secret value and binds it to a sensor policy (e.g., access permission in the allowed geofence). They also implemented and evaluated two abstractions based on two TEEs, ARM TrustZone and x86-based hypervisor. However, they only focus on how to handle sensing value inside the TEEs of the sensing device.

Gilbert *et al.* [3] proposed a method to assure the local sensing data processing and the device owner's privacy in mobile sensing services. To solve these problems, they presented a trusted platform that consists of hardware and software components trusted by the service provider and the device owner. For the integrity of sensing data, the TCB captures a sensing data and signs it with a private key in the TPM. For protecting the device owner's privacy, they separated a sensing app from the trusted core so that the app cannot directly access security-sensitive sensors. However, the size of TCB components including a hypervisor, and a trusted OS, and mobile hardware, is relatively large compared to other related work.

Dua *et al.* [1] proposed a TPM-based trusted sensing platform for the integrity and share of the sensing data. This approach attests the integrity of the platform and sensing data with the TPM and protects the sensing data with the Augmented Broadcast Encryption scheme to share data with multiple parties. However, it is vulnerable to various attacks like the platform integrity because this approach does not leverage the TEE to protect the code base for attestation.

IBM proposed a new truck-tracking solution [10] based on IBM Blockchain [11] and Watson IoT [12]. This solution uses blockchain to store the relevant information captured from sensors placed on trucks. Because the information shared by the blockchain repository can only be altered with consensus from all groups (i.e., logistics companies, producers, and consumers), malicious groups cannot manipulate the sensing data stored in the blockchain. However, this approach merely assures the trustworthiness of the sensing data stored in the blockchain, not one captured from sensors.

Shepherd *et al.* [5] proposed a protocol to secure a TEE-to-TEE communication channel between two remote TEEs. This approach focuses on a problem of one-way trust verification of TEE; that is, a challenger attests the platform integrity of an attester, but not vice versa. To address this problem, the proposed protocol verifies trust uni-directionally or bi-directionally based on remote attestation protocol. The authors also performed formal verification of the proposed protocol with Scyther [13], a protocol verification language.

Weiser *et al.* proposed SGXIO [14] so as to realize the trusted path in the Intel software guard extension (SGX) [15] environment. The key feature of SGX distinguished from other TEEs is that an SGX enclave runs in the user mode. However, higher privileged-layers like a kernel and a hypervisor cannot access code and data protected by the SGX enclave. This feature hinders the realization of the trusted path on top of SGX, because code running in the user mode cannot handle I/O events directly. To overcome this problem, SGXIO leverages a hypervisor to manage I/O peripherals directly and establishes a secure channel between the hypervisor and an SGX enclave.

Although many researchers have proposed related work, there is no practical end-to-end implementation of the trusted sensing for the IoT environment. To address these problems, we present a novel trusted sensing framework for the IoT environment. In this paper, we state a trusted sensing framework for the IoT environment and implement it to show our approach's feasibility.

III. BACKGROUND: ARM TRUSTZONE

Our sensing board relies on ARM TrustZone, which is capable of delivering the TEE for a mobile device. In this section, we state key features of TrustZone to be used for safeguarding the sensing board.

Memory isolation is a key feature of the TrustZone technology. TrustZone can compartment memory by using the TrustZone protection controller (TZPC) and the TrustZone address space controller (TZASC). The TZPC can configure peripherals as secure or non-secure and the TZASC can partition its address range into several regions, which can be configured as secure or non-secure. In addition, the secure OS can prevent peripherals from using them in the Normal world by mapping their addresses into a secure memory region. In our implementation, the address of the serial port (i.e., 0x5000c000) for the sensor connection is mapped into a secure memory region.

The secure monitor call (SMC) instruction triggers a mode switch from the Normal world to the monitor mode. The monitor mode is a gatekeeper to control the mode switch from one world to the other world. The monitor mode software saves the context of the current world and restores the context of the new world. On the other hand, setting the non-secure (NS) bit in the secure configuration register (SCR) to 1 can switch into the Normal world without running a specific instruction. When initializing the secure OS in the Secure world and finishing an SMC instruction, the secure OS sets the NS-bit to 1 in order to switch into the Normal world. Before switching to the Normal world, the secure OS switches to the monitor mode by writing the value (i.e., 0x16) to the current process status register (CPSR).

To pass data from one world to the other world, the monitor mode software plays an interface role between the two worlds. It can transfer data from one world to the other based on world-shared memory through marshalling and unmarshalling data. Our prototype uses the world-shared memory to pass a sensing value, a nonce, and an evidence value.

IV. PROBLEM STATEMENT

A. Problem Overview

Trusted sensing is a protected channel between a sensor in a device and a server-side program. In this section, we identify problems in order to satisfy the trusted sensing and state why they have to be addressed.

First of all, security-sensitive code for the trusted sensing must run in the TEE in order to work as intended. If that code runs in the untrusted execution environment, attackers can compromise a trusted sensing framework. To make a trusted sensing framework trustworthy, code for sensor reading and cryptographic operations must run in the TEE. In addition, the security-sensitive code must be self-contained in the TEE. If leveraging software components (e.g., a driver and a library for a sensor) that reside in the rich execution environment (REE), the TEE can minimize the TEE's TCB size. However, attackers can acquire and craft sensing values in the REE because the REE is the untrusted execution environment. Therefore, the security-sensitive code for the trusted sensing must be self-contained in the TEE.

Table 1: Security requirements for TruSense.

No.	Requirements	Section
G1	Self-contained sensor reading code in the TEE	VI.D
G2	Self-contained cryptographic code in the TEE	VI.D
G3	Secure sensor under the direct control of the TEE	VI.C
G4	Persistent secure storage for cryptographic key	VII.A
G5	Ephemeral secure memory for integrity value	VI.B
G6	Trusted boot based integrity measurement	VI.B
G7	Verifiable TCB and sensing value	VII.B

Second, sensors used for the trusted sensing must be under the direct control of the TEE. If being capable of accessing sensors in the REE, attackers can craft sensing values. Thus the access to the sensors should be dedicated to the TEE.

Third, the trusted sensing framework requires secure storage to safeguard security-sensitive values used during trusted sensing operations. Secure storage is classified into persistent and ephemeral storage for security-sensitive data. The persistent secure storage is used to store long-term data like cryptographic keys, which lasts for a device lifetime. On the other hand, the ephemeral secure storage is secure memory for short-term sensitive data such as the integrity value of the secure chain. Typically, this type of secure storage permits a read operation from the REE (e.g., reading a platform configuration register in a TPM), but restricts an update operation from the REE. In a trusted sensing platform, only the TEE can access both the secure storages but the REE cannot read nor modify data in the secure storages.

Lastly, the integrity of a sensing device's TCB and its sensing value must be verifiable in a server-side program for the trusted sensing. The trusted boot is a commonly used way to measure the security-sensitive code. The trusted boot measures all TCB components and securely manages their integrity values using ephemeral secure storage. The sensor reading code hashes and stores a sensing value in memory inside the TEE. This code also signs the integrity values regarding to the TCB and the sensing values for the integrity verification to be performed by the server-side program.

Table 1 shows the security requirements to address the aforementioned problems. We mark each security requirement where TruSense satisfies the corresponding requirement.

B. Adversary Model

Threats in a sensing device. We assume that all software components running in the REE of a sensing device are insecure. It means that attackers can perform all kinds of attacks against our framework running in user and kernel modes in the Normal world. For example, they can manipulate a sensing value in a kernel-mode driver, user-mode libraries, and other processes. However, we assume that all software components running in the Secure world are trustworthy.

Threats in a communication channel. We assume that attackers can compromise a sensing value transmitted over the network. For example, they can eavesdrop or manipulate a sensing value sent to a cloud service. Moreover, they can forge network packets to deceive the cloud server.

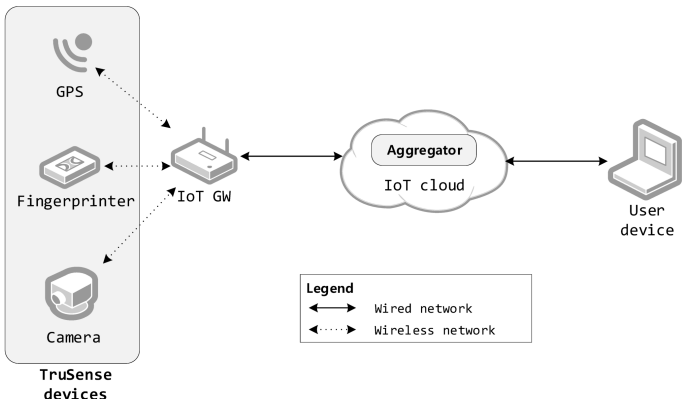


Fig. 1: The TruSense framework for the IoT environment.

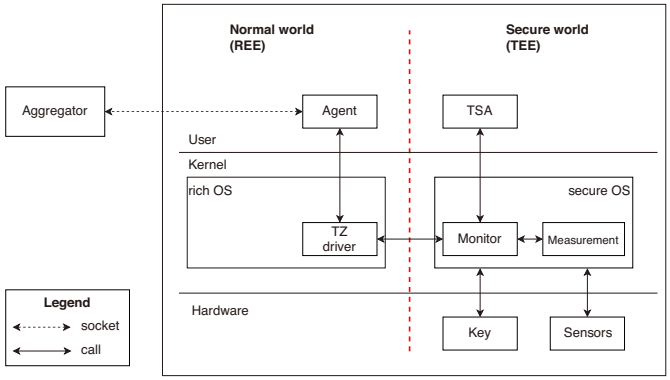


Fig. 2: Overall architecture of the TruSense device.

C. Assumption

Our approach focuses on countermeasures against attacks inside an IoT sensing board. Therefore, we do not consider the manipulation of the signal outside a sensor. In addition, the cloud service is under the control of the cloud provider, and we thereby assume that the cloud service works as intended.

V. TRUESENSE FRAMEWORK

In this section, we propose a trusted sensing framework of the IoT environment, referred as TruSense. We also present the overall architecture of TruSense, and then describe how the components of the TruSense architecture interact with each other to provide a trustworthy service.

Fig. 1 shows the overall TruSense architecture consisting of IoT devices, an IoT gateway, and a cloud service.

TruSense device. A TruSense device is an IoT sensing board that gathers information from a sensor such as a GPS, a camera, or a fingerprint reader, and periodically uploads sensing values to a cloud service. To provide a trustworthy service, the TruSense device has to assure the integrity of the sensor readings. To this end, we leverage the ARM TrustZone extensions to protect the integrity of sensor readings from all other software running on the REE. A trusted application running on the TrustZone-aware OS collects the sensor readings, and signs the sensor readings with the private key of the TruSense device before passing them to the rich OS. An agent running on the rich OS delivers the sensor readings, the signature, and the certificate of the TruSense device to the cloud service. Of the transmitted data to the cloud service, the certificate is used to verify the signature of the sensor readings.

IoT gateway. An IoT gateway is a connection point to transmit sensing data to a cloud service. Typically, IoT devices use various connectivity modules (e.g., Bluetooth low energy and WiFi) in order to connect to the Internet. The used modules depend on the characteristics of IoT devices. With the connectivity modules, the TruSense device sends the sensing data to the cloud service via the IoT gateway. In this paper, we use an off-the-shelf IoT gateway to send the sensing data to the cloud service.

Aggregator. In the proposed architecture, a cloud service, named the aggregator, gathers sensor readings from a lot of TruSense devices, and verifies the integrity of the sensor readings using cryptographic operations. Since the amount of the sensing data dynamically varies in the IoT environment, the elastic resource management is beneficial to the trusted sensing framework. The cloud computing meets this property so that we implement the aggregator on the Google app engine, which is one of the commonly used cloud services. The cloud tenant can use a web browser in a user device to view the aggregated sensing data over an HTTPS channel.

In the proposed framework, all communication channels between the TruSense device and the aggregator are protected by a secure protocol such as the TLS/SSL.

VI. DESIGN OF TRUSENSE DEVICE

In this section, we state how a trusted TruSense device guarantees the integrity and authenticity of sensing data inside the TruSense device.

With the proposed TruSense device, the aggregator can verify the trustworthiness of the sensing data from the trusted IoT device. To this end, the TruSense device leverages the ARM TrustZone extensions as an isolation method between the TEE and the REE, and adopts the trusted boot to produce the TCB measurement. For the trusted sensing, the TruSense device operates the trusted sensing application (TSA) running in the TEE. The TSA reads a sensing value from a sensor dedicated to the TEE and signs the sensing data and the TCB measurement value with the private key of the TruSense device. The agent, which is an application running in the Normal world, uploads the output of the TSA to the aggregator via REST APIs.

Fig. 2 shows the overall architecture of the TruSense device. In the Secure world, the secure OS is the key part for the trusted sensing and plays the central role to protect the TSA for handling sensing data from the untrusted execution environment (i.e., the rich OS). The TSA uses services of the secure OS in order to access secure resources such as a sensor, a key, and a measurement value.

In the Normal world, the agent and the TZ driver run on the rich OS and merely relay values from the secure OS to the aggregator, The agent invokes a system call of the rich OS to the

TZ driver in order to request a command to the secure OS and receive an evidence for the trusted sensing. The communication channel between the agent and the aggregator is safeguarded by the TLS protocol. Because all evidences and sensing data are cryptographically computed in the secure OS, the agent and the TrustZone driver cannot affect the trustworthiness of TruSense.

A. TCB

The TCB is the set of hardware, firmware, and/or software components that are critical to its security of trusted sensing. In the TruSense framework, software TCB components for the trusted sensing are a boot loader, a secure OS, and trusted applications running on the secure OS. To guarantee their trustworthiness, we use the trusted boot regarding the software TCB components, which are commonly used for the integrity of the secure boot chain. We state the detailed explanation of the secure boot and trusted boot in the Section VI.B.

Generally, the trusted boot guarantees the integrity of the boot chain comprised by software components, like the boot loader and the secure OS. In TruSense device implementation, the secure OS image contains the binary image of trusted applications. Therefore, the trusted boot regarding the secure OS image can cover the integrity check of the trusted applications.

On the other hand, the TCB of our work does not contain the agent running on the rich OS because the agent merely sends and receives communication data between the secure OS and aggregator. Even though adversary crafts communication data at the outside of the TCB, the aggregator can be aware of the manipulation through cryptographic operations. Thus, it cannot affect the trustworthiness of the sensing data, so that the agent is not a component of the TCB.

B. Trusted Boot

The secure boot and the trusted boot are commonly used to build up the trust chain of a target device. The key difference between them is the guarantee of the secure state after the boot. That is, the secure boot allows a sensing board to boot only into a secure state, while the trusted boot securely reports on the state of the boot [16]. Our sensing board adopts the trusted boot so that it boots regardless of the TCBs' measurement results (**G6**). Instead, the aggregator verifies the boot state of the sensing board in order to check if sensing values aggregated by the aggregator are valid.

The necessary components to achieve the trusted boot are the root of trust measurement (RTM) and the ephemeral secure memory. The trusted boot starts from immutable RTM code, the 1st boot loader, in an AP. In turn, the 1st boot loader measures the 2nd boot loader, and the 2nd boot loader measures the secure OS. After that, the 2nd boot loader passes the measurement values of the trust chain to the secure OS when calling the entry point function of the secure OS. Finally, the secure OS securely stores them inside the TrustZone-protected volatile memory (**G5**).

C. Secure OS in TrustZone

A secure OS is an operating system that runs in the Secure world. Like general OSes, the key role of the secure OS is to manage the resources under the control of the Secure world. In our approach, the secure OS manages the access to sensors (i.e., a GPS, a fingerprint, and a camera), and allows a trusted application to securely compute an evidence value of a sensing value with a cryptographic key and the TCB measurement (**G3**). Note that the sensing value, the cryptographic key, and the measurement are only accessible in the Secure world.

Of the available open source OSes, such as TLK [17] and OP-TEE [18], we chose Genode [19] as a secure OS in the Secure world. Genode is a microkernel-based OS that features the kernel core isolation from other OS components (e.g., device drivers, a memory manager, and so on) and the small memory footprint. Those features are appropriate to secure OS implementation that requires the small TCB size and the restriction of bug propagation from other components.

To handle various sensors in the Secure world, we customize Genode to include a module that handles a GPS sensor, a fingerprint module, and a camera, which are all connected to the sensor connector on the sensing board. The well-known weakness of microkernel-based OS is the inter-process communication (IPC) overhead, and it is inevitable to our customization. In our approach, we consider that it is the trade-off between security and performance. We adopt the microkernel-based OS to safeguard our customized components for prioritizing the trustworthiness of the proposed approach.

D. Trusted Sensing Application in TrustZone

A TruSense device executes a trusted application, TSA, on the microkernel-based secure OS. The TSA gathers sensing data from a sensor and generates evidences on the sensing data. As a trusted application running in the secure OS, the TSA can read the genuine values of the sensing data (**G1**), the TCB measurement and an attestation key. We explain the attestation key in detail in Section VII.A. To produce the evidences, the TSA hashes a nonce from the aggregator, the TCB measurement, and the sensing data. Then, the TSA signs the hashed value using the attestation key (**G2**). The agent in the REE uploads these evidences from the TSA.

E. Agent in REE

An agent running in the REE relays communication data from and to the aggregator and interworks with the TSA in the TEE via the SMC instruction. The agent secures the REST API-based communication by establishing the HTTPS connection with the aggregator. The TZ driver in the REE executes the SMC instruction when the agent attempts to acquire evidences on the sensing data from the TSA.

VII. TRUSENSE PROTOCOL

A. Attestation Key Distribution

An attestation key is an asymmetric, non-migratable signing key to identify a device uniquely. This key must be signed by a trusted party (e.g., a chip manufacturer or a platform vendor) and its certificate also comes along with the attestation key. The key role of the attestation key is to sign a measurement value of the TCB components when a party (e.g., aggregator) requests

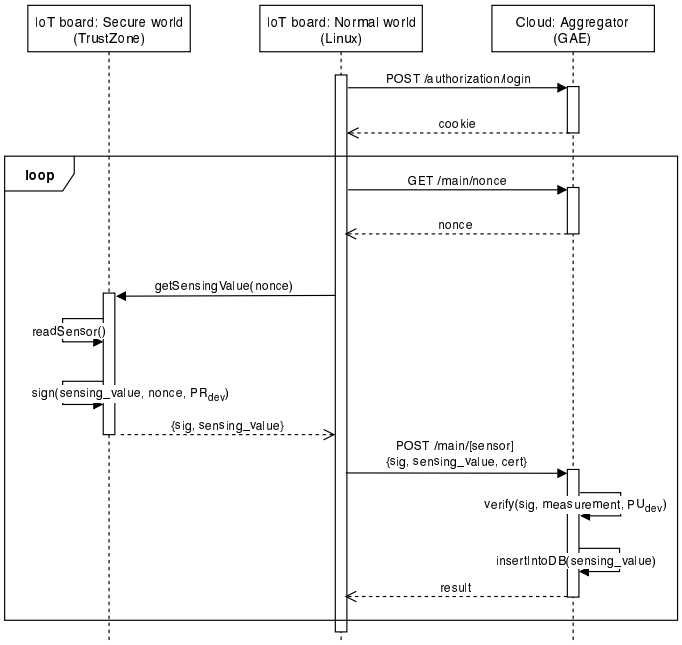


Fig. 3: The UML sequential diagram of the trusted sensing protocol.

an evidence of the platform integrity. For example, a measurement value signed by an attestation key can be used to verify it cryptographically with a corresponding certificate in the party. If the attestation key is compromised, an attacker can impersonate a genuine device that owns the leaked key so that it should be securely managed.

Typically, a device manufacturer fuses a device-specific key, which is a device root key (DRK), into secure area in an AP at the first stage during the production process [20]. Moreover, only a privileged software module like a trusted application in the Secure world can access the DRK. It indicates that the successful verification of the attestation value implies the evidence of performing the sign operation in the Secure world. The device manufacturer creates a DRK certificate including the corresponding public key of the DRK and signs with the device manufacturer's private key. The TruSense device keeps the certificate in the internal storage and sends it along with sensing data to the aggregator in order that the aggregator verifies the evidence of the trusted sensing data.

TruSense adopts the aforementioned secure management of the DRK. Since this management of the DRK meets the requirements of the attestation key, TruSense utilizes the DRK as the attestation key (**G4**).

B. Periodic Trusted Sensing Protocol

To guarantee the trustworthiness of sensing values, we design a periodic trusted sensing protocol with the REST APIs and use the HTTPS protocol to secure the communication between the TruSense device and the aggregator. Fig. 3 is the UML sequence diagram of the proposed protocol.

HTTPS connection establishment. An agent running in a TruSense device initiates and establishes a secure connection over the HTTPS protocol. During the establishment process, the agent verifies the GAE's certificate with the certificate authority (CA) certificate. If the certificate verification is successful, an HTTPS connection is established. Next, the aggregator authenticates the TruSense device with identifiers such as a shared secret, an X.509 certificate, etc. As a consequence of this process, the agent receives a cookie for keeping the authentication state from the aggregator. The agent sends communication data with the cookie for the aggregator's session management.

Nonce request. The agent requests a nonce to the aggregator to keep the freshness of the signature and prevent the replay attack. The aggregator generates a random value (`nonce`) and keeps it into a session variable so as to verify the signature later. Next, the aggregator returns the nonce to the agent.

Trusted sensing value retrieval. Upon receiving the nonce, the agent invokes the SMC instruction with the nonce via the kernel in the Normal world. From this point, the control flow is switched to the Secure world. First, the TA obtains a sensing value (`sensing_value`) from a sensor dedicated to the Secure world. In turn, the TA signs the TCB measurement, the sensing value, and the nonce with the attestation private key ($\texttt{PR}_{\texttt{dev}}$). Lastly, the TA returns the sensing value and the signature to the agent, and the agent transmits them along with the device's certificate (`cert`) to the aggregator.

The parameters passed to the sign operation have the specific meanings for security guarantee. The measurement value of the secure OS indicates that the intact trusted OS running in the Secure world retrieves a sensing value from a sensor only accessible from the Secure world. Moreover, the sign operation with the private key under the direct control of the Secure world guarantees that the signature was generated in the Secure world, because the private key is only accessible in the Secure world.

Sensing value verification. After receiving the signature and the sensing value from the TruSense device, the aggregator verifies `cert` from the agent that contains the attestation public key ($\texttt{PU}_{\texttt{dev}}$). After that, the aggregator verifies the signature with a known-good measurement value of the secure OS and the attestation public key (**G7**). If the verification succeeds, the aggregator inserts the sensing value into the database. Finally, the aggregator returns the result of the sensing value verification to the TruSense device.

The successful verification means that a sensing value from a sensing board is obtained by the intact secure OS (i.e., the known-good measurement value) and came from an identifiable and legitimate sensing board (i.e., the attestation public key corresponding to the TruSense device).

VIII. IMPLEMENTATION

TruSense device. We created our own TruSense device for the trusted sensing using a TrustZone-enabled AP. This board equips an ARM Cortex-A8 1 GHz processor, a WiFi module, a JTAG port, a battery connector, an LED indicator of the Secure world, a sensor connector based on UART, and so on.

Figs. 4(a) and 4(b) show the frontside and backside of the sensing board, respectively. Fig. 4(c) represents the TruSense device that connects with a GPS and a battery. This board can also connect with other UART-connected sensors (e.g., a finger-

print and a camera). The size of the current TruSense device can be minimized more by removing parts (e.g., a JTAG port and a serial port) only used for the debugging purpose.

We ported two OSes into the Secure and Normal worlds. The secure OS running in the Secure world is Genode 15.02 [19], an open source microkernel-based OS. To achieve our goals, we customized `core` of Genode to manage the measurement of TCB components like Genode. We also added a Genode driver, `uart`, that reads sensing values from sensors via the UART interface. The rich OS running in the Normal world is Linux 2.6.35. An application running in the Normal world communicates with the aggregator in the GAE via the HTTPS protocol and the Genode OS by running the SMC instruction.

Aggregator cloud service. To deploy the aggregator in the cloud, we chose GAE, which is a productive, flexible platform-as-a-service (PaaS) by Google. Like other PaaSes, it supports several popular programming languages and databases. We implemented the aggregator with the Node.js and the MySQL database delivered by the GAE. Fig. 5 is the screenshot of the aggregator cloud service to show the sensing data aggregated from the GPS-connected sensing boards.

Limitation. Without the support of the chip vendor, it is infeasible to construct the RTM and the persistent secure storage in an AP. Thus, we assume that the RTM and the persistent secure storage reside in the AP. To realize this assumption, the trusted boot in our implementation starts from the 2nd boot loader instead of the RTM. We also hard-code the DRK, which is supposed to be placed in the persistent secure storage, in the image of the secure OS.

IX. EVALUATION

In this section, we evaluate the TruSense device from the aspect of the performance overhead, power consumption, and TCB size.

Usually, the TEE deployment contributes to the REE's performance overhead because some privileged operations cause a trap to the TEE. However, since our TruSense device is dedicated to perform the trusted sensing, the secure OS in our TruSense device does not need to trap them. Therefore, only the SMC instruction causes a trap to the TEE so that the performance impact on the REE is negligible. Moreover, the data copy overhead between the REE and the TEE also leads to the TEE environment's performance overhead. To avoid this performance impact, we use shared memory to hand over data to the other side. Since there is no copy operation between both worlds, we can avoid the performance impact due to data transfer. Because of the reasons, we do not evaluate the performance impact on the REE regarding world switching and data copy.

Another evaluation factor of the ARM TrustZone based device is the power consumption due to the introduction of the secure OS. He Sun et al. proposed a trusted one-time password (OTP) [21] using ARM TrustZone and a Freescale i.MX53 development board, which embeds the same AP as the TruSense device. In [21], they mentioned that the power consumption regarding world-switching goes up a little bit. Since the TruSense device and the Trust OTP device equip the same AP, they have a similar tendency to consume the power during world-switching. Therefore, we do not evaluate this factor in this paper.

Table 2: Elapsed time and power consumption of core operations. Avg. of 100 runs.

Comparison factors	**Key pair generation**		**Sign**	
	RSA	ECC	RSA	ECC
Elapsed time (ms)	5,140.4	11.4	72.0	16.2
Power consumption (mW)	1,251.47	5.50	20.32	6.56

A. Selection of Cryptographic Algorithm

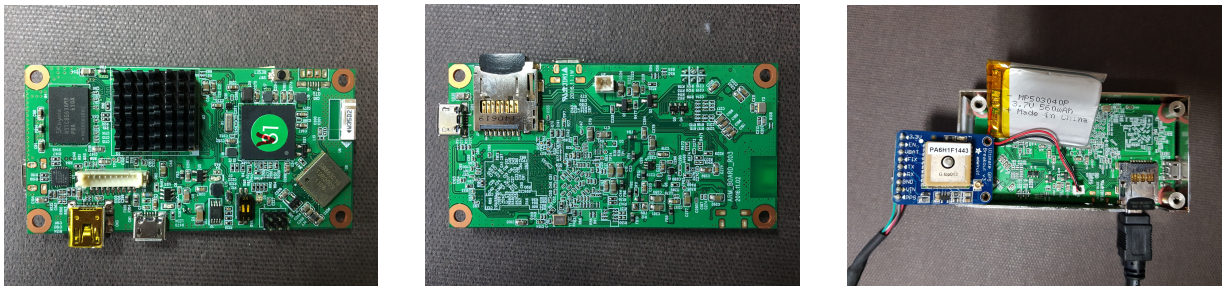
Typically, it is well known that ECC is more efficient than RSA from the processing time and power consumption standpoint. Table 2 shows the result of the processing time and power consumption based on RSA (2048-bit) and ECC (secp256k1) in our TruSense device. As shown in Table 2, the ECC sign operation is approximately 4.4 times faster and 3.09 times more power-efficient. In the case of the key generation, the elapsed time of ECC outperforms one of RSA by approximately 450 times. Typically, the RSA key generation is less efficient due to its operation mechanism; that is, the RSA key generator must find two large random prime numbers. If it does not match this property, the RSA key generator continues to use a random number generator until the generated keys become large random prime numbers. This process leads to the nondeterministic key generation time and the high power consumption.

Despite the RSA's inefficiency, in some cases the TruSense framework has to introduce the RSA algorithm inevitably. For example, a cloud tenant can introduce a hardware security module (HSM) for the strong key protection in the cloud. A HSM-manufacturer [22] charges extra payments in order to add the ECC algorithm into the HSM. In this case, the cloud tenant should use the RSA algorithm to reduce operating costs.

To overcome the RSA's weakness, reducing the key size of RSA can be a good option. According to algorithms, key size and protocols report (2018) [23], using 80-bit encryption (i.e., 1024-bit RSA key) may be appropriate to keep data secret for a very short space of time. The sensing data meets the condition for the use of the short key size because it has the short lifetime for sensing data transfer from the TruSense device to the aggregator. Even though the RSA key generation leads to performance overhead, it is tolerable because it is rarely executed only in the first phase. We represented the performance enhancement of using the 1024-bit RSA key pair in our previous work [6].

B. Selection of Maximum CPU Frequency

The minimum and maximum CPU frequencies of the sensing board are set at 400 MHz and 1 GHz, respectively. Generally, the lower clock frequency has the benefits of lower power consumption. According to this rule, it is likely appropriate to set the maximum CPU frequency as lower as possible because the low power consumption is one of the important metrics for IoT devices. However, it is correct in the case of running an I/O-bound workload. Fig. 6 and Table 3 illustrate the power consumption of the ECC cryptographic operations by varying clock frequency. As expected, the sensing board running at 1 GHz fin-



(a) (b) (c)

Fig. 4: The description regarding the pictures above goes here: (a) The front side of the sensing board, (b) the back side of the sensing board, and (c) the GPS-connected sensing board.

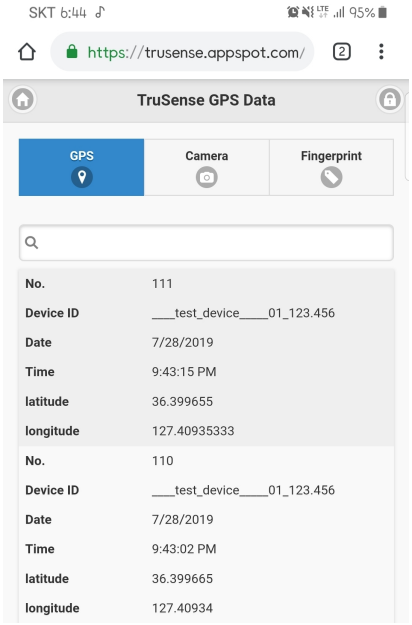


Fig. 5: The screenshot of the aggregator cloud service running in the GAE.

Table 3: Sum of the power consumption (mW) of the operations regarding the AP clock speed. Sum of 100 runs.

Clock speed	Key pair generation	Sign
400 MHz	743.27	840.74
1 GHz	549.65	655.78

ishes the cryptographic operations earlier as well as shows lower power consumption. Therefore, we set the maximum CPU frequency at the AP's highest one (1 GHz). The dynamic frequency scaling of the sensing board adjusts the maximum frequency, depending on the workload.

C. Performance Comparison with Hypervisor-based Trusted Sensing

In our previous work [6], we proposed TGVisor, a hypervisor-based trusted geolocation framework using the TPM for the cloud environment. In this subsection, we compare both trusted sensing frameworks from the perspective of the performance and TCB size. Table 4 shows the comparison of both trusted sensing frameworks.

Key generation. The execution time in TruSense is $5,140.4$ ms, while one of TGVisor is $7,794.4$ ms. To generate an RSA key pair, an RSA key generator uses a random number generator. TruSense leverages a jitter-based random number generator, a virtual file system plugin where only the TEE is accessible. On the other hand, TGVisor calls `TPM_Random` that generates 1-byte random value using the true random number generator inside the TPM. Because a TPM 1.2 module used in TGVisor uses the old-fashioned mechanism like the Low Pin Count bus, the performance degradation by the TPM's RNG is inevitable. Thereby, the larger the RSA key size is, the more severe the performance impact on the key generation process is.

Sign with TPM. Since TruSense does not embed the TPM, we cannot measure this factor. On the other hand, TGVisor takes 437.0 ms to sign a GPS value with the `TPM_Quote2` operation, which is a TPM sign operation to sign platform configuration register (PCR) values with an RSA private key placed in the TPM. Due to the same reason to the TPM key generation, the TPM's low performance causes the long latency of the TPM sign operation.

Sign without TPM. The execution time of TruSense is 72.0 ms, whereas one of TGVisor is 90.0 ms. There is only a slight, negligible difference, depending on instruction set architectures (ISAs) where TruSense and TGVisor are implemented. Thus, we conclude that the performance of the main processors of TruSense and TGVisor (i.e., 1 GHz of an ARM Cortex-A8 and 2.3 GHz of an AMD Turion P520) make this slight difference (18 ms).

TCB size. The TCB size is one of the key factors of a security system to show how secure it is. Generally, a system can benefit from the small TCB size because it has the small attack surface. To count the LoC of target OSes, we use CLoC [24]. The TCB size of TruSense and TGVisor is $27,539$ LoC and $8,311$ LoC, respectively. This LoC difference is due to the characteristic of the TEEs, Genode and XMHF [25]. Genode is a general-purpose, secure OS for the Secure world, while XMHF is a security-purpose, lightweight hypervisor. That is, XMHF does not contain features for the general hypervisor such as resource management so that it can have the small TCB.

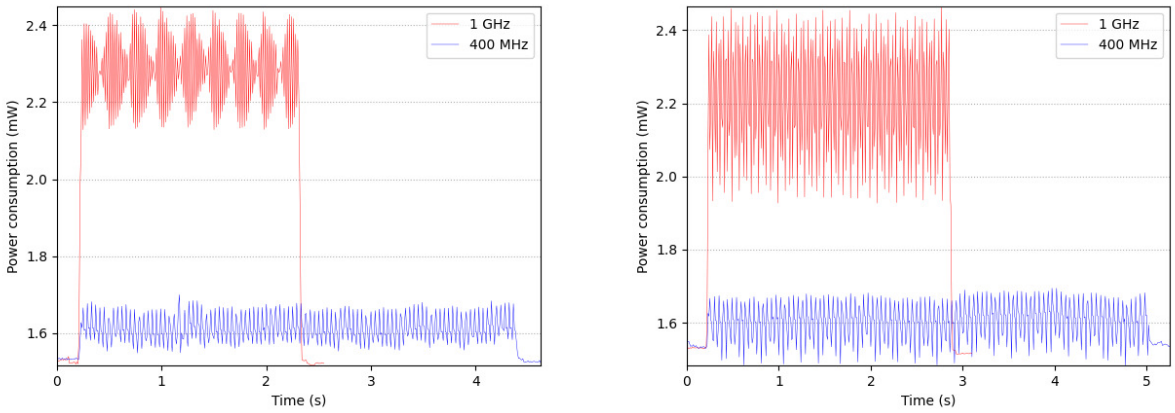


Fig. 6: Power consumption of 100 runs regarding the ECC cryptographic operations: (a) ECC key generation and (b) ECC sign operation.

Table 4: Performance comparison of trusted sensing-related operations between ARM TrustZone and hypervisor.

Factors	TEE type		Description
	TrustZone	Hypervisor	
ISA	ARM	AMD x64	Used ISA
Key generation	5,140.4	7,794.4	Generate a RSA-2048 key pair in the TEE (ms)
Sign operation w/ TPM	-	437.0	Sign a GPS value with a RSA-2048 private key present inside the TPM (ms)
Sign operation w/o TPM	72.0	90.0	Sign a GPS value with a RSA-2048 private key generated in the TEE (ms)
TCB size	27,539	8,311	The line of code (LoC) of the TCB

X. CONCLUSION AND FUTURE WORK

The trusted sensing framework is an essential feature to maintain a mission-critical IoT environment securely. To build up the trust of the IoT environment, we designed and implemented the end-to-end trusted sensing framework for the IoT environment and showed the feasibility of our approach. We also evaluated and compared our framework with the hypervisor-based trusted sensing framework.

One challenging problem of our work is that *there is no consideration regarding a key management scheme for the IoT environment, which consists of billions of IoT devices and cloud services*. With the CA-based key management scheme, the CA can pose a single point of failure and result in a bottleneck for handling many key management operations. As future work, we will introduce a blockchain-based key management system to meet requirements such as the self-management property for the IoT environment.

REFERENCES

[1] A. Dua, N. Bulusu, W.-C. Feng, and W. Hu, "Towards trustworthy participatory sensing," in *Proc. USENIX HotSec*, Aug. 2009, p. 8.
[2] H. Liu, S. Saroiu, A. Wolman, and H. Raj, "Software abstractions for trusted sensors," in *Proc. ACM MobiSys*, June 2012, pp. 365–378.
[3] P. Gilbert, L. P. Cox, J. Jung, and D. Wetherall, "Toward trustworthy mobile sensing," in *Proc. ACM HotMobile*, Feb. 2010, pp. 31–36.
[4] M. Bartock *et al.,* "Trusted geolocation in the cloud: Proof of concept implementation," *Publication NISTIR*, vol. 7904, Dec. 2015.
[5] C. Shepherd, R. N. Akram, and K. Markantonakis, "Establishing mutually trusted channels for remote sensing devices with trusted execution environments," in *Proc. ACM ARES*, Aug. 2017, pp. 1–10.
[6] S. Park, J.-J. Won, J. Yoon, K. H. Kim, and T. Han, "A tiny hypervisor-based trusted geolocation framework with minimized TPM operations," *J. Systems Software*, vol. 122, pp. 202–214, Dec. 2016.
[7] European Telecommnucations Standards Institute (ETSI), "Technical specification. Cyber security for consumer internet of things," [Online] Available: https://www.etsi.org/deliver/etsi_ts/103600_103699/103645/01.01.01_60/ts_103645v010101p.pdf, 2019.
[8] Google, Google app engine, [Online] Available: https://cloud.google.com/appengine/docs/, 2019.
[9] J. Oh, J. Park, S. Park, and J. J. Won, "Taaas: Trustworthy authentication as a service based on trusted path," in *Proc. IEEE CLOUD*, June 2016, pp. 27–34.
[10] K. Lewis, Truck-tracking solution protects cargo with IoT and blockchain, [Online] Available: https://www.ibm.com/blogs/internet-of-things/iot-tracking-solutions-blockchain/.
[11] IBM, IBM Blockchain, [Online] Available: https://www.ibm.com/blockchain/.
[12] IBM, Watson Internet of things, [Online] Available: https://www.ibm.com/internet-of-things.
[13] CISPA, Scyther tool, [Online] Available: https://people.cispa.io/cas.cremers/scyther/. 2019-7-30.
[14] S. Weiser and M. Werner, "SGXIO: Generic trusted i/o path for intel sgx," in *Proc. ACM CODASPY*, Mar. 2017, pp. 261–268.
[15] Intel, Intel(R) Software Guard Extensions SDK Developer Reference for Windows OS, [Online] Available: https://software.intel.com/sites/default/files/managed/41/58/sgx-sdk-developer-reference-for-windows.pdf, 2017.
[16] D. Challener, K. Yoder, R. Catherman, D. Safford, L. V. Doorn, *A Practical Guide to Trusted Computing*. IBM Press, 2007.
[17] Trusted Little Kernel, [Online] Available: http://nv-tegra.nvidia.com/gitweb/?p=3rdparty/ote_partner/tlk.git;a=tree.
[18] Open Portable Trusted Execution Environment, [Online] Available: https://www.op-tee.org/, 2018.
[19] Genode - Genode Operating System Framework, [Online] Available: https://genode.org/.

[20] Samsung Electronics, KNOX Platform Security, [Online] Available: https://developer.samsung.com/tech-insights/knox/platform-security.
[21] H. Sun, K. Sun, Y. Wang, and J. Jing, "Trustotp: Transforming smartphones into secure one-time password tokens," in *Proc. ACM CCS*, Oct. 2015, pp. 976–988.
[22] nCIPHER, nShield Connect HSMs, [Online] Available: https://www.ncipher.com/products/general-purpose-hsms/nshield-connect.
[23] ECRYPT-CSA, Algorithms, key size and protocols report (2018), [Online] Available: http://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf.
[24] CLOC, Count Lines of Code, [Online] Available: http://cloc.sourceforge.net.
[25] A. Vasudevan *et al.,* "Design, implementation and verification of an extensible and modular hypervisor framework," in *Proc. IEEE SP*, May 2013, pp. 430–444.

Sungjin Park received the B.S. degree from Inha University in 2002, the M.S. degree from POSTECH in 2005, and the Ph.D. degree in Computer Science, Korea Advanced Institute of Science and Technology (KAIST), South Korea in 2017. He is a Senior Researcher at the Affiliated Institute of Electronics and Telecommunications Research Institute (ETRI). His research interests include cloud computing and system security.

Jaemin Park received the B.S. degree from Handong Global University in 2004, and the M.S. and Ph.D. degrees from KAIST, South Korea in 2006 and 2019 respectively. He is a Senior Researcher at the Affiliated Institute of ETRI. His research interests include cloud computing and network security.

Jisoo Oh received B.S. and M.S degrees from the Electrical and Computer Engineering, Sungkyunkwan University, in 2013 and 2015 respectively. She is a Senior Researcher at the Affiliated Institute of ETRI. Her research interest includes software exploitation.