Towards Efficient Fine-Grained Access Control and Trustworthy Data Processing for Remote Monitoring Services in IoT

Yaxing Chen, Wenhai Sun, *Member, IEEE*, Ning Zhang, *Member, IEEE*, Qinghua Zheng, *Member, IEEE*, Wenjing Lou, *Fellow, IEEE*, and Y. Thomas Hou, *Fellow, IEEE*

***Abstract*— As an important application of the Internet of Things, many remote monitoring systems adopt a device-to-cloud network paradigm. In a remote patient monitoring case, various resource-constrained devices are used to measure the health conditions of a target patient in a distant non-clinical environment and the collected data are sent to the cloud backend of an authorized health care service for processing and decision making. As the measurements involve private patient information, access control and trustworthy processing of the confidential data become very important. Software-based solutions that adopt advanced cryptographic tools, such as attribute-based encryption and fully homomorphic encryption, can address the problem, but they also impose substantial computation overhead on both client and server sides. In this paper, we deviate from the conventional software-based solutions and propose a secure and efficient remote monitoring framework, called SRM, using the latest hardware-based trustworthy computing technology, such as Intel SGX. In addition, we present a robust and lightweight "heartbeat" protocol to handle notoriously difficult key revocation problem. We implemented a prototype of the framework for SRM and show that SRM can protect user data privacy against unauthorized parties, with minimum performance cost compared to existing software-based solutions.**

***Index Terms*— Remote patient monitoring, Internet-of-Things (IoT), fine-grained access control, secure hardware, trusted computing.**

Manuscript received August 17, 2018; accepted November 21, 2018. Date of publication December 6, 2018; date of current version April 10, 2019. This work was supported by the National Key Research and Development Program of China under Grant 2016YFB1000303, in part by the Innovative Research Group of the National Natural Science Foundation of China under Grant 61721002, in part by the Innovation Research Team of Ministry of Education, under Grant IRT_17R86, in part by the National Science Foundation of China under Grant 61502379, Grant 61532015, and Grant 61672420, in part by the Project of China Knowledge Center for Engineering Science and Technology, in part by the China Scholarship Council under Grant 201606280105, and in part by the U.S. National Science Foundation under Grant CNS-1446478 and Grant CNS-1443889. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Jean-Luc Danger. *(Corresponding author: Yaxing Chen.)*

Y. Chen and Q. Zheng are with the School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China. (e-mail: cyx.xjtu@gmail.com; qhzheng@mail.xjtu.edu.cn).

W. Sun is with the Department of Computer and Information Technology, Purdue University, West Lafayette, IN 47907 USA (e-mail: whsun@purdue.edu).

N. Zhang is with the Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, MO 63130 USA (e-mail: zhang.ning@wustl.edu).

W. Lou and Y. T. Hou are with the Department of Computer Science, Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA 24061 USA (e-mail: wjlou@vt.edu; thou@vt.edu).

Digital Object Identifier 10.1109/TIFS.2018.2885287

I. Introduction

REMOTE monitoring is one of the killer applications of the Internet of Things (IoT) system. In a remote patient monitoring scenario, health care providers (HCPs) are allowed to monitor the health conditions of a patient outside the conventional clinical environment, e.g. at the patient's home. The measurements are collected in real time from various IoT devices, for example, user activities from audio and video streaming, biometrics such as weight, blood pressure, heart rate via wearable devices on patients' bodies or sensors installed in the room and then sent to the HCP for further functional processing. Instead of maintaining their individual computing infrastructures, nowadays HCPs adopt the public cloud to provide such remote health care services [1].

Due to the private and sensitive nature of the measured information, there is a crucial need for effective and flexible access control and secure data processing to protect user data against unauthorized access while keeping the usability and functionalities of the system. The patient can permit an authorized HCP to access data types based on the offered service. For instance, a cardiovascular monitoring service may need to access the information of electrocardiogram and heart rate. At the same time, the data processing should be secure against unauthorized parties and adhere to the intended service functions.

Much work has been done in the literature to address this problem. For example, attribute-based encryption (ABE) [2]–[5] is a well-known technique used in a variety of applications to achieve scalable, secure, fine-grained access control. On the other hand, privacy-preserving data processing can be realized by secure multi-party computation [6], fully homomorphic encryption (FHE) [7]. However, such pure crypto-based solutions typically involve complex crypto operations. It is likely unable for the battery-powered and extremely resource-constrained devices at the patient's side to afford such computationally-intensive cryptographic operations. Another challenge is the realization of on-demand key revocation and privacy-preserving data protection. The former typically requires a cumbersome large-scale key update as well as storage re-encryption; the latter is usually considered to be prohibitively expensive if we target generic computations, rather than a special class of computation.

In this work, we propose a secure and efficient remote monitoring framework in the context of IoT, which was named as SRM. Compared to the software-based solutions that exploit cryptographic primitives as building blocks, SRM leverages the hardware-based trusted computing technology, such as Intel SGX to protect user data privacy and enable secure computations over sensitive data. Specifically, in remote patient monitoring, assuming a current smart home IoT platform, e.g. Samsung SmartHome [8], we set up a trusted broker in the home gateway to provide data encryption, remote attestation and key management on behalf of the user (i.e., patient). On the cloud server, access control enforcement and data processing are performed in a trusted execution environment (TEE) protected by secure hardware. Our proposed approach represents a major departure from existing software-based solutions. Moreover, it is very efficient due to the use of secure hardware. For one reason, SRM only adopts symmetric encryption, such as AES[GCM] [9] to encrypt/decrypt data. For another reason, it carries out the monitoring service (i.e., HCP) functions that could be arbitrary constitutions over plaintext data, rather than encrypted ciphertext data.

On the other hand, there is a significant challenge that we need to address before delivering the claimed secure and efficient properties of SRM. By our design, the device secret keys on the untrusted cloud server only can be used in the enclave (SGX term for TEE) and the trustworthy executions of the access control enforcement and monitoring processing are guaranteed by SGX enclave functions. Suppose that a monitoring service maliciously initializes an enclave with tempered function codes, it will be detected during remote attestation. However, stronger attackers, such as OS and VM hypervisor, may launch packet-drop attack to compromise the system. For example, it is expected that the trusted broker can explicitly inform the enclave to erase device secret keys to revoke the access permission of the service. However, a malicious OS may selectively ignore such request and help the revoked services to continue reading the user's data. Worse still, a compromised monitoring application may fail to invoke corresponding enclave functions to respond the revocation request. In order to solve this problem, we propose a "heartbeat" protocol. In a nutshell, we force the enclave of the revoked monitoring service to be unavailable if it does not receive a valid heartbeat from the trusted broker after the defined time window.

To summarize, we make the following contributions:

- Building upon recent development of secure processor, we propose a practical secure remote monitoring framework, called SRM, which offers fine-grained access control and privacy-preserving data processing on user information. Compared to existing software-based solutions that rely on cryptographic primitives, the proposed system offers rich functionality while incurring small performance overhead.
- We propose a novel "heartbeat" protocol to address the limitation of Intel SGX architecture, where it is possible for the untrusted cloud server or a monitoring application to selectively drop network traffic to prevent the user from further controlling the enclave upon initial remote attestation. The "heartbeat" protocol allows revocation of previous entrusted key materials in the enclave.
- We formally define the proposed framework and implemented a prototype with supporting network communication stacks on both the trusted broker and monitoring service sides. Besides, we performed abundant experiments and numerical analysis to thoroughly evaluate the framework.

II. SGX Background

SGX [10], [11] is the latest Intel instruction extensions and allows the host application to reserve a protected memory region as trusted execution environment (TEE), called *enclave*, so that sensitive application operations can run inside securely against privileged system software,[1] e.g. OS kernel, VM hypervisor. In addition, SGX provides two other important functions, *storage sealing* and *remote attestation*. Storage sealing allows the enclave to protect its data on the untrusted persistent storage; remote attestation enables a distant entity to check the integrity of the newly generated enclave, including the internal state, code, etc. Should the verification be successful, the entity is able to establish an authenticated secure channel and deliver its secrets into the enclave. As an essential building block of our framework, next we will provide more technique details about the remote attestation function.

A. Remote Attestation

To enable this function, Intel SGX platform provisions a special enclave called quoting enclave. When a challenged enclave remotely attests to an entity, it needs first to locally attest to the quoting enclave as follows. First, the challenged enclave sends a unique signed structure known as *report* to the quoting enclave, which contains the two enclave's identities, some meta-data and a MAC. An enclave's identity is measured by the values of two registers: MRENCLAVE, which is a SHA-256 digest of an internal log of all activities done while the enclave is built; and MRSIGNER, which is a hash of the public key of the party who signs the enclave prior to distribution. The MAC is calculated using a unique report key derived from the Root Seal Key that is hardcoded when the Intel SGX enabled processor is manufactured. After the *report* is received, the quoting enclave then verifies it by re-computing the MAC over the underlying data of the *report* with a report key recovered by itself. If the two MAC values are equal, it shows that the challenged enclave is indeed an enclave running on the same hardware platform with the quoting enclave since they have the same report key. In other words, the firmware and hardware of the challenged enclave are trustworthy. Next, the quoting enclave generates a new signed structure call *quote* by re-signing the underlying data of the *report* using the Intel Enhanced Privacy ID (attestation key) [12], [13], which is an anonymous group signature scheme implemented by Intel to overcome the privacy concerns in standard asymmetric signing schemes where a small number

[1]The trusted computing base (TCB) of SGX only comprises the CPU and several privileged enclaves.

of keys are used across the life of the platform. Finally, the $quote$ is delivered to the entity who in turn transfers it to the Intel Attestation Service (IAS) for validation. In principle, any verifier that possesses the group public key can verify the $quote$.

In order to formally describe our proposed scheme, we model the secure SGX-enabled hardware as a black-box program similar to [14] and [15]. For simplicity, we only outline the main functionalities and interfaces related to our work.

B. SGX Model

A secure hardware platform $\mathbb{HW}$ for a class of probabilistic polynomial time (PPT) programs $\mathcal{P}$ consists of following algorithms. Additionally, it has a read-only memory $state$ for recording data initialized during setup and a table denoted by T, consisting of enclave state tuples indexed by enclave handles, to manage the internal states of the loaded programs.

1) $params \leftarrow Setup(1^{\lambda})$*:* This algorithm takes in a security parameter λ. It generates the remote attestation key pair (sk_{quote}, pk_{quote}), and records them in $state$. It outputs pk_{quote} as public parameters $params$.

2) $hdl_p \leftarrow Load(params, p)$*:* This algorithm takes in a stateful program $p \in \mathcal{P}$ and global public parameters $params$. It creates an enclave, loads p into the enclave, and generates a handle hdl_p that identifies the enclave running p. Finally, it initializes the entry $T[hdl_p] = \emptyset$ and returns hdl_p.

3) $out \leftarrow Run(hdl_p, in)$*:* This algorithm takes in an enclave handle hdl_p and inputs in. It runs p on in, sets $T[hdl_p]$ to be the updated state, and returns the output out.

4) $quote \leftarrow Quote(hdl_p, in)$*:* This algorithm takes in an enclave handle hdl_p and inputs in. It first executes p on in to get out and updates $T[hdl_p]$ accordingly. Then, it returns the attestation $quote = (md_{hdl_p}, tag_p, in, out, \sigma)$, where, md_{hdl_p} is the meta-data associated with hdl_p, tag_p is a cryptographic hash of p that can be used by the challenger to verify the program running inside the enclave, and σ is a cryptographic signature produced using sk_{quote} on $(md_{hdl_p}, tag_p, in, out)$.

5) $b \leftarrow QuoteVerify(params, quote)$*:* This algorithm takes in the public parameters $params$ and an attestation $quote$. It first verifies the signature σ with pk_{quote} in $params$ and then compares the enclave program tag tag_p with the pre-computed tag of the trustworthy program p. It returns $b = 0$ for any verification failure. Otherwise, it returns $b = 1$.

C. Limitations

Intel SGX, however, is known to be vulnerable to various physical and software attacks, for example, side-channel attacks [16]–[18] including cache-timing attack, power analysis attack, branch shadowing attack, etc. Besides, a malicious OS can launch DoS attack [19] to disrupt the enclave function as it is still in charge of the underlying resource allocation. Furthermore, the inputs and invocation of an enclave function cannot be enforced [20], which means that a host application may not properly invoke the well-designed enclave functions. Thus, this limitation of Intel SGX architecture on the intuitive

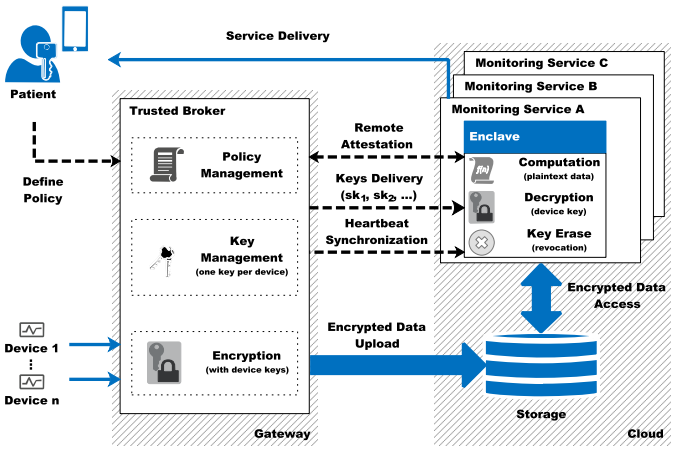

Fig. 1. The proposed framework for remote patient monitoring and protocol flows.

SGX-based remote patient monitoring system allows the monitoring service to continue accessing the user data by dropping off the explicit revocation command from the trusted broker.

III. Problem Formulation

A. System Model

A remote patient monitoring system in our design consists of a user (i.e., patient) and various monitoring services (i.e., health care provider) as shown in Fig. 1. At the user's end, multiple devices, either wearable or physically fixed in the room, are deployed to measure the health conditions of the user. The health care information collected from the monitoring devices are sent to a user-controlled gateway, where a trusted broker program executes to manage the access policy for each subscribed monitoring service, device secret keys and data encryption. Then it uploads encrypted data as per device to the cloud storage. Monitoring services, outsourced by hospitals, skilled nursing facilities, disease research centers, etc., have respective specialties in health-related data analysis, assessment and recommendations to the users. These services in our system set up SGX enclaves to perform the computation involving sensitive user information. To this end, a monitoring service host application first needs to request the corresponding device secret keys from the user gateway after a successful remote attestation. Then the service enclave loads the intended ciphertext of user data from cloud storage and securely process them after decryption. In order to revoke an existing subscripted service of the user, a robust "heartbeat" protocol is running between the trusted broker and the service enclave. Normally, the enclave will securely erase all the acquired device secret keys when it receives a revocation command along with a heartbeat signal. Any exceptional situation will cause the enclave out of service.

B. Threat Model

We assume that the monitoring environment containing IoT devices, the gateway and communication channels between them is trustworthy. In addition, we do not trust the cloud

including applications, OS kernels, VM hypervisor, etc., except for CPU and enclave internals, which is consistent with the security of SGX. Thus, we, in general, exclude the relevant physical and software attacks on SGX in this paper. However, we do consider the challenging issue of key revocation under the packet-drop attack, where the malicious OS or compromised monitoring application may selectively cast away the revocation request.

C. Design Goals

Our proposed framework aims to achieve the following design goals. With respect to system performance and functionalities,

- ***Scalability*:** Our scheme should be scalable and allow the user to subscribe as many monitoring services as he/she needs in practice.
- ***Efficiency*:** The overhead of proposed security mechanisms should be minimal.

Pertaining to security, our framework mainly realizes the following goals,

- ***Confidentiality of user data and secret keys*:** It is expected that the measured user data are well protected when stored and processed in the cloud and the corresponding secret keys will not be disclosed to unintended parties.
- ***Trusted monitoring functional processing*:** The data processing operations of a monitoring service in the cloud should be verifiable and comply with the prescribed service agreement.
- ***Fine-grained data access control*:** An authorized monitoring service can only access the data types defined by the user.
- ***Heartbeat robustness*:** The proposed "heartbeat" protocol should be resistant to some abnormalities and/or active attacks, such as network failure, replay attack, and data forging.

IV. Secure and Efficient Remote Monitoring Framework

A. Main Idea

In SRM, it is expected that the private user data should be securely processed and also compliant with the subscribed monitoring service. In order to achieve this, we leverage Intel SGX to create an enclave for the user and put all the sensitive information and computation into the enclave. By remote attestation and computation environment isolation by the enclave, we can ensure that the enclave is faithfully and securely performing the expected monitoring functions.

In addition, the user should be able to enforce access control policy for the subscribed service over his/her outsourced data and revoke the access permission of the unsubscribed service. To achieve this, a unique random secret key is assigned to each monitoring device that outputs a specific health-related data type. Data confidentiality can be realized by using the key to encrypt the relevant type of data. Further, the user can also control which monitoring service can access what types of user data by providing the corresponding device secret keys. Intuitively, in order to revoke an existing service and prevent it from further accessing the user data, we may re-encrypt the data type that was allowed for the target service with updated device keys and redistribute these keys to the affected services. Obviously, this method incurs considerable computation and communication overhead, and cannot revoke the access permission promptly, which is very important for a real-time remote monitoring system. The user can also choose to explicitly send a revocation command to the enclave to destroy all assigned device keys, but it will fail if the malicious OS or compromised monitoring service host application intercepts this request. To solve this challenging issue, we present a "heartbeat" protocol in SRM. The core idea is to send a periodical signal from the user side to retain the service enclave's vitality and force the enclave to erase all the assigned device keys if it receives an explicit revocation command along with the signal. If the enclave does not receive a valid signal during a predefined time window, it is forced to be out of service.

Last, we automate the scheme for the user by executing a trusted broker program in the user-side gateway device to enable various critical security functions, such as encryption, key management, attestation, etc.

B. Notations

We use the syntax of the SGX model $\mathbb{HW}$ defined in Section II to describe the oracles provisioned by the secure hardware platform on the monitoring service's side. The loaded stateful enclave program, denoted by $\mathbb{E}_{App}$, includes following internal states: $(vk_{sp,sign}, sk_{sp,sign})$ for recording the signature key pair of the monitoring service, sk_{comm} for recording the shared communication key between the trusted broker and the enclave, L_{keys} for recording device keys that are allowed to access by the enclave, hb_state for recording the latest enclave state updated by receiving valid heartbeats, c' for recording the maximum counter in the received heartbeats. The monitoring functional computation f and heartbeat freshness $threshold$ are predefined and hardcoded in $\mathbb{E}_{App}$.

We also model a stateful program, denoted by $\mathbb{F}_*(\cdot)$, to represent main functionalities of the trusted broker on the user's side, which consists of five subroutines: $\mathbb{F}_{RA}(\cdot)$ for remote attestation, $\mathbb{F}_{KM}(\cdot)$ for key management, $\mathbb{F}_{DE}(\cdot)$ for data encryption, $\mathbb{F}_{PM}(\cdot)$ for policy management, $\mathbb{F}_{HB}(\cdot)$ for heartbeat emission. In addition, it includes following internal states: ACL for recording access control policies, DKL for recording secret device keys, CKL for recording shared communication keys, $DHKL$ for recording Deffie-Hellman parameters, $(vk_{u,sign}, sk_{u,sign})$ for recording the signature key pair of the user. The cryptographic hash $tag_{\mathbb{E}_{App}}$ of the trustworthy enclave program $\mathbb{E}_{App}$, and the heartbeat frequency hb_freq are predefined and hardcoded in $\mathbb{F}_*(\cdot)$.

Furthermore, E denotes an authenticated symmetric key encryption scheme, which consists of $KeyGen(\cdot)$, $Enc(\cdot)$ and $Dec(\cdot)$ algorithms. S denotes an existentially unforgeable signature scheme, which consists of $KeyGen(\cdot)$, $Sign(\cdot)$ and $Verify(\cdot)$ algorithms. The main notations are summarized in Table I.

TABLE I
MAIN NOTATIONS

Notation	Description
$\mathbb{HW}$	The SGX hardware platform functional model.
$\mathbb{E}_{App}$	The SGX enclave program run by the monitoring service.
$hdl^{*}_{\mathbb{E}_{App}}$	The handle for a SGX enclave running $\mathbb{E}_{App}$. $*$ can be $alive$ indicating that the enclave is active for functional processing, $stale$ indicating that it is out of service.
$\mathbb{F}_{*}(\cdot)$	The trusted broker functional model. $*$ can be KM for key management, Enc for encryption, PM for policy management, HB for heartbeat mechanism.
sk_{comm}	the secure communication key shared between the trusted broker and the monitoring service enclave.
$(vk_{u,sign}, sk_{u,sign})$	the signature key pair generated by the trusted broker for authentication.
$(vk_{sp,sign}, sk_{sp,sign})$	the signature key pair generated by the monitoring service for authentication.
sk_{dev}	The symmetric secret key for device dev.
ct_{dev}	The ciphertext of the data file bound to device dev encrypted with sk_{dev}
L_{keys}	A key list, which contains secret device keys that can be accessed by a monitoring service.

C. Framework Description

Fig. 2 illustrates our proposed scheme, which comprises a *pre-processing phase* and six protocols: **Init**, **Upload**, **Subscribe**, **HBSetup**, **Monitoring**, **Unsubscribe**. Relevant main functions of $\mathbb{F}_{*}(\cdot)$ and $\mathbb{E}_{App}$ in each protocol are formally defined.

Pre-Processing Phase: The user bootstraps and configures the trusted broker in the gateway and registers a family of monitoring devices $\mathcal{D} = \{dev_i | i = 1, \cdots, m\}$ to the trusted broker. Additionally, a family of monitoring services $\mathcal{R} = \{sp_j | j = 1, \cdots, n\}$ run $params \leftarrow \mathbb{HW}.Setup(1^{\lambda})$ to initialize their own SGX hardware platforms. Relevant platform parameters $params$ are output as system public parameters.

Init. The trusted broker generates a unique ID $dev_i[id] \leftarrow \{0,1\}^{\lambda}$ and a unique secret key $sk_{dev_i} \leftarrow E.KeyGen(1^{\lambda})$ for each registered device in $\mathcal{D}$, and records the tuple $(dev_i[id], sk_{dev_i})$ in its device key list DKL. It also generates a unique signature key pair $(vk_{u,sign}, sk_{u,sign}) \leftarrow S.KeyGen(1^{\lambda})$ for the user. At last, the trusted broker outputs the registered device IDs $U_{dev} = \{dev_i[id] | i = 1, \cdots, m\}$ and the verification key $vk_{u,sign}$ as public parameters in the system. Formally,

$$\mathbb{F}_{KM}: \ input(\text{“}init\text{”}, \mathcal{D}, 1^{\lambda}) \ ; \ output(U_{dev}, vk_{u,sign}).$$

Meanwhile, every involved monitoring service in $\mathcal{R}$ generates a global unique ID $sp_j[id] \leftarrow \{0,1\}^{\lambda}$, and a unique signature key pair $(vk_{sp_j,sign}, sk_{sp_j,sign}) \leftarrow S.KeyGen(1^{\lambda})$. The service IDs and the corresponding verification keys $U_{sp} = \{(sp_j[id], vk_{sp_j,sign}) | j = 1, \cdots, n\}$ are output as system public parameters.

Upload. When the trusted broker receives the data msg collected from the user and ambient environment by a monitoring device $dev_i[id] \in U_{dev}$, it looks up DKL to obtain the entry $(dev_i[id], sk_{dev_i})$ and then outputs a corresponding cipthertext $ct_{dev_i} \leftarrow E.Enc(sk_{dev_i}, msg)$. Formally,

$$\mathbb{F}_{DE}: \ input(msg, dev_i[id]) \ ; \ output(ct_{dev_i}).$$

The ciphertext is subsequently uploaded by the trusted broker to the cloud.

Subscribe. This protocol interacts between the trusted broker and the monitoring service.

When a user subscribes to a service $sp_j \in \mathcal{R}$, he/she first defines and adds a unique access rule $(sp_j[id], L_{dev})$ in the access control list ACL. The device ID list $L_{dev} = \{dev_i[id]\}$ designates the devices that can be accessed by the service. The trusted broker then generates a random number $sk_{u,dh} \leftarrow \{0,1\}^{\lambda}$ as the Diffie-Hellman secret and a corresponding public parameter $pk_{u,dh}$. The tuple $(sp_j[id], pk_{u,dh}, sk_{u,dh})$ is recorded in $DHKL$. Finally, it outputs the $pk_{u,dh}$ along with the signature $\sigma_{pk_{u,dh}} \leftarrow S.Sign(sk_{u,sign}, pk_{u,dh})$. Formally,

$$\mathbb{F}_{PM}: \ input(\text{“}policy\ add\text{”}, L_{dev}, sp_j[id]); \\ output(pk_{u,dh}, \sigma_{pk_{u,dh}}).$$

At the same time, the service initializes a dedicated enclave instance $hdl_{\mathbb{E}_{App}}$ for the target user by loading the pre-defined program $\mathbb{E}_{App}$ into the enclave, i.e., $\mathbb{HW}.Load(params, \mathbb{E}_{App})$.

Next, the service performs $\mathbb{HW}.Quote(hdl_{\mathbb{E}_{App}}, \text{“}attest\text{”}, 1^{\lambda}, pk_{u,dh}, \sigma_{pk_{u,dh}})$ to begin the remote attestation interaction with the trusted broker. Specifically, the program running in the enclave receives the $pk_{u,dh}$, $\sigma_{pk_{u,dh}}$ as inputs in. It first authenticates the trusted broker by executing $S.Verify(vk_{u,sign}, \sigma_{pk_{u,dh}}, pk_{u,dh})$. If passed, it then generates its own Deffie-Hellman secret and corresponding public parameter $(sk_{e,dh}, pk_{e,dh})$, and derives the shared communication key sk_{comm} with the received $pk_{u,dh}$. After that, the program runs $S.Sign(sk_{sp,sign}, pk_{e,dh})$ to produce a signature $\sigma_{pk_{e,dh}}$ and assembles it with $pk_{e,dh}$ as outputs out. Note that we assume the signing key $sk_{sp,sign}$ of the service is delivered to the enclave in a secure way (one possible method is through local attestation). By locally attest to the platform quoting enclave, the service finally obtains a unique attestation $quote$ of its established enclave, i.e., $quote = (meta_{hdl_{\mathbb{E}_{App}}}, tag_{\mathbb{E}_{App}}, in, out, \sigma)$. Formally,

$$\mathbb{E}_{App}: \ input(\text{“}attest\text{”}, 1^{\lambda}, pk_{u,dh}, \sigma_{pk_{u,dh}}); output(quote).$$

After receiving the attestation $quote$, the trusted broker invokes the $QuoteVerify$ algorithm for verification. If passed, it then parses the out to obtain $(pk_{e,dh}, \sigma_{pk_{e,dh}})$ and runs $S.Verify(vk_{sp_j,sign}, \sigma_{pk_{e,dh}}, pk_{e,dh})$ to authenticates the monitoring service. If passed, it derives the shared communication key sk_{comm} using the recorded $sk_{u,dh}$ and the received $pk_{e,dh}$, and adds an entry $(sp_j[id], sk_{comm})$ to its communication key list CKL. Any failure during the above processes will result in the abort of attestation. Formally,

$$\mathbb{F}_{RA}: \ input(\text{“}verify\text{”}, quote, sp_j[id]) \ ; \ output(b_1).$$

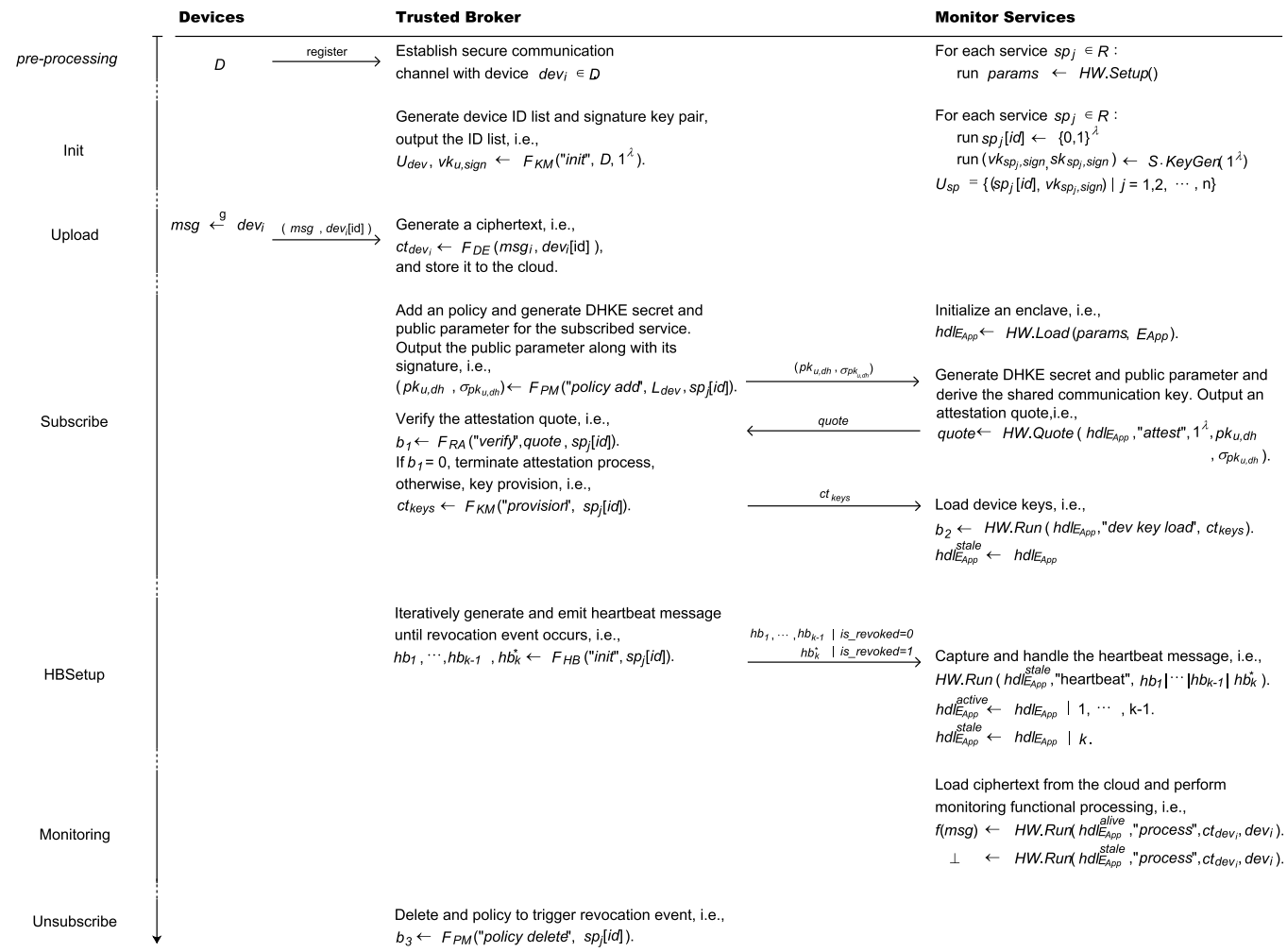

Fig. 2. The proposed remote monitoring scheme.

Provided that the attestation is successful, i.e $b_1 \neq 0$, the trusted broker assembles a device key list $L_{keys} = \{sk_{dev_i}\}$ for the target service by looking up ACL and DKL, and encrypts it with the corresponding sk_{comm} in CKL by running $E.Enc(sk_{comm}, L_{keys})$. Formally,

$$\mathbb{F}_{KM} : input(\text{“}provision\text{”}, sp_j[id]) \; ; \; output(ct_{keys}).$$

The ciphertext ct_{keys} is then delivered to the enclave instance.

On the other side, within the enclave, the program decrypts the ciphertext ct_{keys} by running $E.Dec(sk_{comm}, ct_{keys})$, and recovers the device key list L_{keys}. Formally,

$$\mathbb{E}_{App} : input(\text{“}dev\ key\ load\text{”}, ct_{keys}) \; ; \; output(b_2).$$

Remark: By design, the successful execution of this protocol, i.e., $b_2 \neq 0$, guarantees the enclave instance $hdl_{\mathbb{E}_{App}}$ can obtain the device secret keys, but it still cannot be used to handle user's data since its internal variable hb_state has not yet been updated to keep its freshness by the "heartbeat" synchronization mechanism. We mark such state of the enclave as inactive, denoted by $hdl^{stale}_{\mathbb{E}_{App}}$.

HBSetup. This protocol interacts between the trusted broker and the monitoring service.

Suppose that the remote attestation succeeds, the trusted broker will create a dedicate thread to periodically emit heartbeats to the service. The frequency hb_freq is preset by the system. Currently, our framework supports two kinds of heartbeats: *non-revocation heartbeat*, denoted by hb, triggers to update the freshness of the service enclave, indicating the sustainability of the service; and *revocation heartbeat*, denoted by hb^*, triggers to erase all internal states of the service enclave, indicating the termination of the service. In general, during a whole serving period of t, the trusted broker will generate $k = hb\_freq \times t$ heartbeats, among which the first $k-1$ heartbeats are *non-revocation heartbeat* while the last one is *revocation heartbeat*.Formally,

$$\mathbb{F}_{HB} : input(\text{“}init\text{”}, sp_j[id]); \; output(hb_1, \cdots, hb_{k-1}, hb^*_k).$$

On the service's end, it also creates a dedicated thread to capture those heartbeats sent by the trusted broker and transfers them to the enclave, in which the program parses each heartbeat and updates its internal states accordingly. Formally,

$$\mathbb{E}_{App} : input(\text{“}heartbeat\text{”}, hb_1|\cdots|hb_{k-1}|hb^*_k); \; output(ENCLAVE\_STATUS).$$

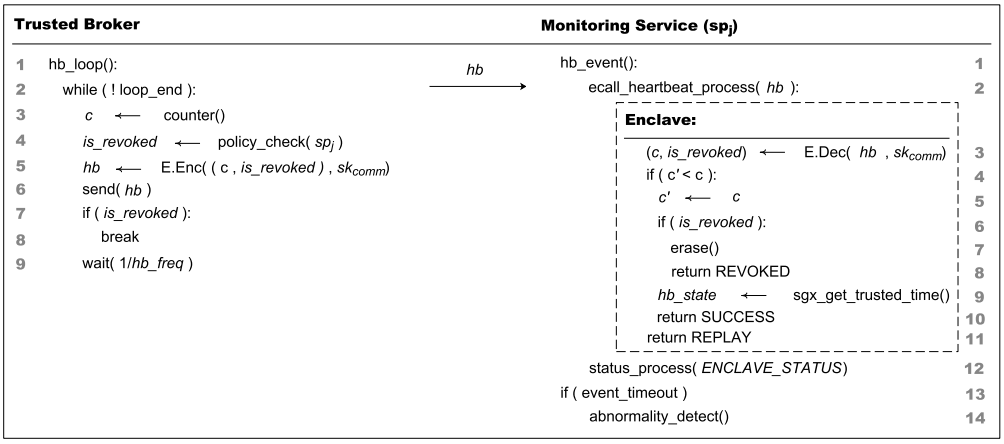

Fig. 3. The proposed "heartbeat" protocol interacting between the trusted broker and the monitoring service.

Normally, if the enclave successfully processes a *non-revocation heartbeat*, i.e., $ENCLAVE\_STATUS = SUCCESS$, the enclave's state will be updated to date, such that the enclave can properly perform subsequent monitoring functions over user's data. We mark such state of the enclave as active, denoted by $hdl^{active}_{\mathbb{E}_{App}}$. As for other situations including various active attacks, we will give detail discussions in Section IV-D.

Monitoring. A monitoring service sp_j loads a ciphertext ct_{dev_i} of the device dev_i from the cloud storage and transfers it to its enclave instance. The program running in the enclave first executes an assert that verifies the freshness of the hb_state, which will be discussed in Section IV-D. If passed, it retrieves sk_{dev_i} from L_{keys}, runs $E.Dec(sk_{dev_i}, ct_{dev_i})$, performs the pre-defined monitoring function f over the plaintext, and outputs the result $result$. Otherwise, it denies to serve for the current protocol execution and output $\perp$. Formally,

$$\mathbb{E}_{App} : input(\text{“}process\text{”}, ct_{dev}, dev) \; ; \; output(result | \perp).$$

Remark: This is not a one-time protocol. As long as the enclave instance is opened by the monitoring service, it should be executed on demand to constantly load ciphertexts from the cloud storage and deliver service to the user.

Unsubscribe. When a user opts to cancel a monitoring service sp_j, the trusted broker looks up its access control list ACL and deletes the corresponding entry of sp_j. Once the termination state of the dedicated "heartbeat" thread is set, it then deletes the entry $(sp_j[id], sk_{comm})$ from its communication key list CKL. Formally,

$$\mathbb{F}_{PM} : input(\text{“}policy\ delete\text{”}, sp[id]) \; ; \; output(b_3).$$

D. Heartbeat Synchronization Protocol

Fig. 3 shows the proposed "heartbeat" protocol, which runs between the trusted broker and the monitoring service.

1) On the Trusted Broker Side: We implemented a loop function $hb\_loop()$ to enable periodical heartbeat emission. Each iteration represents a heartbeat cycle. We also use a variable $loop\_end$, initialized as $false$, to control the on-off switch of the loop (line 2). During each heartbeat cycle, it first generates a monotonically increased positive number c (line 3). We implemented a $counter()$ function to achieve that. It then checks the current revocation status of the target monitoring service and stores the status in a boolean variable $is\_revoked$ (line 4). Concretely, the invoked $policy\_check(sp_j)$ function looks up ACL for the entry of the service. If no entry exists, it returns $true$; Otherwise, $false$. Next, the loop function generates a heartbeat hb by using the shared communication key sk_{comm} of the target service to encrypt c and $is\_revoked$ and then sends the hb to the monitoring service in the cloud (line 5 and 6). If the service has not been revoked, i.e., $is\_revoked = false$, the current process will be suspended for a defined time period $1/hb\_freq$ before entering the next cycle, which guarantees a stable heartbeat frequency. Otherwise, it will exit the loop and stop sending heartbeat signals (line 7-9).

2) On the Monitoring Service Side: We implemented an event response function $hb\_event()$ to monitor heartbeats (line 1). Once it receives a heartbeat hb from the trusted broke, it transfers the hb to its enclave for processing by calling the pre-defined ECALL function $ecall\_heartbeat\_process(hb)$ (line 2). Within the enclave, it decrypts the hb with the shared communication key sk_{comm} to recover the number c and the revocation status $is\_revoked$ (line 3). Next, it checks whether c is larger than the number c', which is maintained by the enclave to record the latest received c (line 4). Note that c' is initialized to be -1. If $c \leq c'$, the enclave returns the state of REPLAY to the host monitoring service (line 11). Otherwise, the enclave updates c' with the newly received c (line 5). Then, it checks the revocation status (line 6). Provided that the service were revoked, i.e., $is\_revoked = true$, the memory for storing the obtained secret keys from the trusted broker will be erased, and the state of REVOKED will be returned (line 7 and 8). Otherwise, the enclave updates the freshness state hb_state with a tuple $(cur\_time, nonce)$ generated by a SGX library function *sgx_get_trusted_time()* and returns the state of SUCCESS to the host monitoring service (line 9 and 10). Technically, the cur_time is a trusted time-stamp in seconds relative to a time source indicated by the $nonce$. Timestamps can be compared only if the related $nonce(s)$ are equal. Note that Intel use the implemented

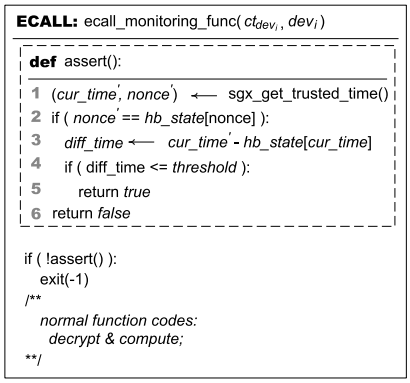

Fig. 4. The pseudo-code of an enclave monitoring function and the freshness assert algorithm.

specific capability in the chipset on the platform to ensure that the time-stamp is trustworthy.

The monitoring service bases the returned status from its enclave, i.e., REPLAY, REVOKED and SUCCESS, to do further processing (line 12). Specifically, REPLAY indicates that the enclave suffers from the replay attack, which may alarm relevant parties to perform some defensive actions. The monitoring service will continue its service to the user on receiving a SUCCESS status, or terminates its service if REVOKED status is received. Notably, if the monitoring service doesn't receive a heartbeat from the trusted broker for a defined time period $event\_timeout$ and the REVOKED status has not yet been set, it indicates the system may suffer from abnormalities, such as network failure. This can trigger the detection function (line 13, 14), which is out of the scope of this paper.

Further, as Fig. 4 shows, we need to enhance enclave monitoring functions by inserting an assert before normal function codes are executed, which checks the freshness of the hb_state. First, it invokes the SGX library function $sgx\_get\_trusted\_time()$ to obtain a tuple $(cur\_time', nonce')$ (line 1). Suppose that the $nonce'$ matches the $hb\_state[nonce]$, i.e., using a same time source to generate the trusted time-stamps, it computes the difference $diff\_time$ between the cur_time' and the $hb\_state[cur\_time]$ (line 2,3). If $diff\_time$ is less equal than a pre-defined time window named $threshold$, then it returns true (line 4,5), meaning that the enclave is fresh and that the subsequent codes can be properly executed. Otherwise, it returns false (line 6) and thus the monitoring function directly exits.

One non-trivial issue is how to set the value of $threshold$, which is a trade-off between the timeliness of revocation and the robustness of mechanism. Provided that the $threshold$ were very large compared to the time interval of two adjacent heartbeats, i.e., $1/hb\_freq$, the mechanism can be robust to temporary network failure. However, it may postpone the revocation time and thus cause information leak. For example, when the last *revocation heartbeat* is not received by the enclave because of either OS or monitoring service compromise, it can continue processing the user's data until the time window $threshold$ runs out. On the contrary, if the $threshold$ is close to the time interval, it can enforce to response revocation event in time but may be vulnerable to the network failure, resulting in bad Quality of Service (QoS). We provide a detailed discussion in Section VI-B.

Remark: We allow that keys in the enclave can be flushed to the hostile storage medium in case servers shut down, using the storage sealing functionality provided by the SGX platform. Correspondingly, the related key reloading functions must also perform the freshness checking assert before their functional codes are executed, which can avoid the situation where a revoked service continues processing the user data by reloading previously stored keys.

V. Correctness and Security

A. Correctness

The remote monitoring scheme SRM is correct if for all monitoring devices $dev_i \in \mathcal{D}$ of a user and all monitoring services $sp_j \in \mathcal{R}$, the following two statements hold: 1) Once the user properly subscribes to a monitoring service sp_j, the probability that the user receives an incorrect result of the pre-agreed function defined in the service agreement is negligible; and 2) after the user properly unsubscribes to a monitoring service sp_j, the probability that the service can keep processing the user data is negligible.

Built on the proposed "heartbeat" protocol, the correctness of our scheme can be guaranteed by the follows. In the case of receiving a valid heartbeat in the defined time window, if the carried revocation indicator $is\_revoked$ is 0, the freshness hb_state of the enclave will be updated to date, which allows the service to continue accessing the user data. Otherwise, the access permission of the service will be revoked by erasing all the assigned secret keys in the enclave. Should the heartbeat were not properly received by the enclave during the defined time window, the abnormality, due to either network failure or the intentional drop off of the packet by the malicious OS or compromised service host application, will be detected. As a consequence, the hb_state becomes stale, which fails the freshness check assert and thus disables the critical enclave monitoring functions towards data processing. Note that enclave codes can be audited during remote attestation, such that the above designed execution flow of the enclave is enforced.

B. Security

What follows is a detailed discussion on the security properties that SRM achieves with regard to security goals defined in Section III-C.

1) Confidentiality of User Data and Secret Keys: This property is satisfied by both software-based symmetric encryption schemes, such as AES[GCM], and the used secure hardware TEE function, i.e. Intel SGX enclave.

When outside the enclave, the user information msg_i collected from various monitoring devices dev_i are encrypted using respective device keys sk_{dev_i}. The generated ciphertexts ct_{dev_i} are stored in the cloud. Subsequently, as in the Subscribe protocol, we utilize the remote attestation to establish a secure communication channel between the trusted broker and the enclave. By exchanging Diffie-Hellman public parameters ($pk_{u,dh}$, $pk_{sp,dh}$) and cross verifying their signatures

$(\sigma_{pk_{u,dh}}, \sigma_{pk_{sp,dh}})$, the established channel encrypted with the negotiated shared communication key sk_{comm} can not only guarantee the confidentiality of its escorted data but also defend against key access by enclaves of other unauthorized parities. For example, an active attacker cannot pretend a valid monitoring service to establish a communication channel with the trusted broker through its own constructed enclave. As a result, the relevant device keys can be securely provisioned into the enclave initialized by an authenticated monitoring service. As only the authenticated enclaves obtain the device keys, the encrypted user data can only be decrypted inside these enclaves for functional processing. On the other hand, though the communication key and assigned device keys can be stored in the hostile storage medium, they are encrypted using an enclave-specific key when outgoing the enclave. Thus, the confidentiality of the data and relevant keys are realized in this work as long as the security of SGX is guaranteed.

2) Trusted Monitoring Functional Processing: This property is guaranteed by the remote attestation mechanism of Intel SGX.

During this process, the enclave will generate a $quote$ that includes a cryptographic hash $tag_{\mathbb{E}_{App}}$ of the program $\mathbb{E}_{App}$ running inside the enclave. With this received program tag, the trusted broker on behalf of the user can verify the integrity and correctness of critical monitoring functions that take user private data as input. Thus, the user can be assured of the trustworthy execution of the subsequent data processing and its compliance with the subscribed service agreement.

3) Fine-Grained Data Access Control: We use different device-wise keys to encrypt each data type associated with this device. Thus, the user is able to generate and maintain a straightforward but fine-grained access control policy by explicitly regulating what types of data of the devices can be accessed by the monitoring service.

To achieve this, the trusted broker maintains an access control list ACL and a device key list DKL. In the ACL, an item $(sp_j[id], L_{dev})$ indicates that the monitoring service sp_j can access which devices (represented by a device ID list L_{dev}), while the DKL records the corresponding key sk_{dev_i} of each device dev_i. By assembling these two lists, the trusted broker is enforced to only deliver relevant device keys to enclaves run by different monitoring services.

4) Heartbeat Robustness: We leverage the proposed "heartbeat" protocol to support efficiently and effectively subscribe and unsubscribe an monitoring service. Its robustness relies on the following security properties of the protocol.

- ***Non-forgeability.*** An active adversary may want to construct a valid heartbeat hb to break the protocol. A forged *non-revocation heartbeat* (i.e., $is\_revoke = 0$) can help a revoked monitoring service to continue processing user data; an artificial *revocation heartbeat* (i.e., is_revoke=1) can stop a monitoring service from processing user data even if it has not yet been revoked. In our protocol, the heartbeat is transmitted through the established secure communication channel. More concretely, each heartbeat message is encrypted using a secure authenticated symmetric encryption scheme with the shared communication key sk_{comm}. As the key is only accessed by the trusted broker and the authenticated monitoring service enclave, no adversaries can construct a valid heartbeat. Thus, the protocol is considered to be non-forgeability.
- ***Replay attack resistance.*** An active adversary may record and replay the intercepted *non-revocation heartbeats* to allow a revoked monitoring service to continue processing data. However, we use a monotonically increased number c to maintain the message order. It is expected that c in the newly received heartbeat should be greater than the stored c' in the enclave. Otherwise, the replay attack can be detected.
- ***Network packet-drop attack resistance.*** On one hand, an active adversary who compromises the OS may cast away heartbeats sent to the enclave, and thus causes denial of service. To defend against this, the monitoring service will start abnormality detection if heartbeats were not received after a pre-defined time period $event\_timeout$. Similarly, such a mechanism can also prevent packet loss by network failure. On the other hand, a compromised monitoring service (host application) may selectively discard the *revocation heartbeat* to allow itself to keep processing the user data. Even though the monitoring service does not know whether or not the current heartbeat brings the revocation command, i.e., $is\_revoked = 1$, since the heartbeat message is encrypted, it may exploit other information like service agreement to infer the *revocation heartbeat*. In this case, the freshness assert guarantees that the enclave is still not available for use.

VI. Implementation and Evaluation

A. Implementation

Based on the preliminary work, we implemented an upgraded prototype[2] in C using the Intel SGX SDK 2.1 for Linux, which supports network communication instead of simulation by stub function calls. The prototype is tested on a SGX enabled Intel NUC7i5BNH platform that runs an Intel Kaby Lake i5-7260U processor at 2.20GHz (Turbo frequency can reach to 3.40GHz) with 8 GiB of RAM and Ubuntu 16.01 operating system. Due to the lack of an Intel license required in the release mode, we opted to compile the code using g++ in the debug mode.

Network Module: We leverage the C socket programming to implement relevant network communication stacks at both the trusted broker and monitoring service sides. In particular, to facilitate data transmission between the two entities, we define a uniform data structure pkg_t for various network packets and implemented two auxiliary functions, i.e., $pkg\_serial(\cdot)$ and $pkg\_deserial(\cdot)$, to serialize and/or de-serialize these packets. Besides, we opt to build the network module upon a connectionless network protocol, i.e., UDP. It avoids the back-and-forth connection setup delays, flow control, and retransmission caused by the TCP protocol.

[2]The new version project is available to access through the GitHub via the following link: https://github.com/yxChen1990/SGXEnabledAccess.git

The implication of this design choice is that we allow packet loss during network transmission.

As for other main function modules, i.e., remote attestation, heartbeat, key management, seal secrets, policy management and data processing, please refer to [21] for more details.

B. Evaluation

Next, we give a comprehensive evaluation on SRM. In particular, we try to answer following questions.

1) How is computation performance when using Intel SGX?
2) How is the scalability of SRM in terms of access control?
3) What is the cost by introducing the "heartbeat" mechanism?
4) Given a known heartbeat frequency hb_freq, whether there exists an optimal setting of the enclave freshness threshold $threshold$ under the Internet communication environment?

1) Constant Time Protocols: Majority of protocols in our framework, such as, *pre-processing*, Setup, Subscribe, HBSetup and Unsubscribe, are only executed once in their lifetime. We view such protocols as constant time protocols even though some of them include complex interactions and computations. For example, the Subscribe needs multiple network communications between the trusted broker and the monitoring service and time-consuming verification for remote attestation. Other protocols, i.e., Upload and Monitoring, by design will be executed for multiple times and their wall-clock time per execution depend on the length of inputted plaintext or ciphertext.

2) Performance by SGX Enclave: In this experiment, we aim to measure the performance penalty when using Intel SGX. Specifically, we implemented a 128bit AES[GCM] scheme to demonstrate the additional cost by SGX through evaluating its performance. In the real world, data collected by different monitoring devices varies greatly. For example, a heart-rate sensor may send a 1-byte data while a footage of an activity monitor with a code rate of 4933$kbps$ will need about 616KB frame data per second. To see how the proposed system works under such various conditions, we enable the trusted broker to encrypt files in different sizes. In particular, we chose three file sizes, i.e., 1B, 1KB, and 10KB. In each defined file size, we are also interested in the performance with various number of files since some applications, such as machine learning algorithms, may need to deal with a large number of files. Fig. 5 illustrates the performance of the implemented AES[GCM] scheme. We use the baseline to represent the same implementation without Intel SGX. It can be observed that Intel SGX is more suitable to process (encryption and decryption) small-sized files, i.e., 1 B and 1 KB, where it only imposes a negligible performance overhead. On the other hand, large-sized files, e.g. 10KB, will introduce more performance penalty as the file number increases.

Note that Intel SGX SDK provides a closed-source trusted cryptographic library named sgx_tcrypto, which includes some well-known cryptographic primitives. In particular, it also provides two AES implementations, i.e., Rijndael 128bit-GCM and Rijndael 128bit-CTR. We can choose to use this native

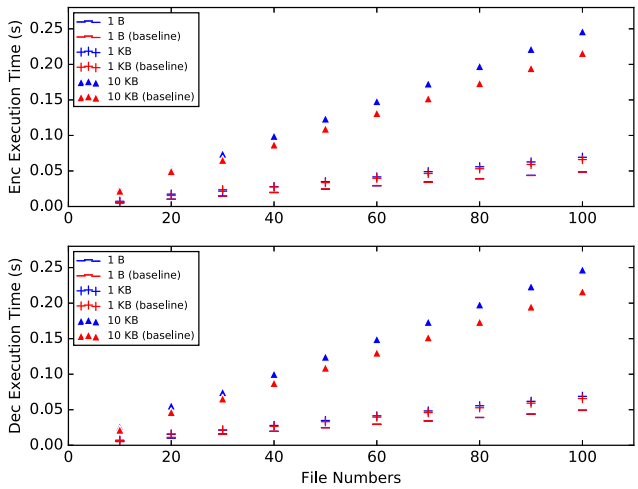

Fig. 5. The computation performance of the implemented AES[GCM] scheme w and w/o Intel SGX.

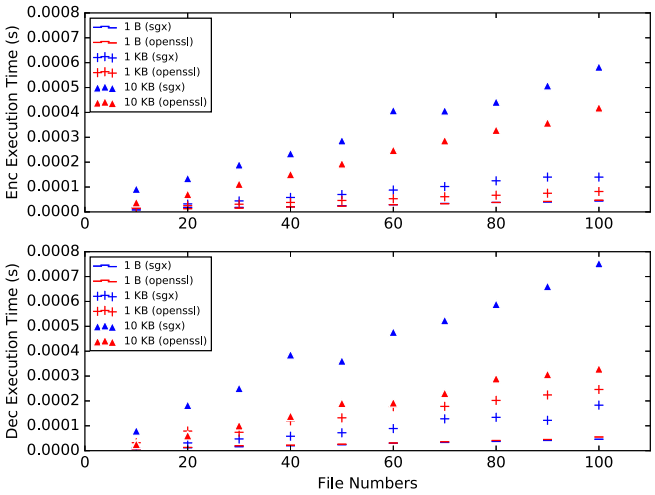

Fig. 6. The computation performance of the off-the-shelf SGX AES[GCM] scheme.

128bit AES[GCM] function to provide the message confidentiality and integrity. Its performance is shown in Fig. 6. Besides those stated conclusions above, we can also learn that the SGX implemented AES[GCM] scheme achieves industry-level performance comparable with the commonly used openssl crypto library, especially for the small-size files.

3) Scalability of SRM in Terms of Access Control: On behalf of the user, a trusted broker is established in the gateway to control the access of his/her devices by multiple monitoring services. On one hand, for a newly subscribed monitoring service sp_j, the trusted broker only needs to add a tuple (sp_j, L_{dev}) in ACL to regulate its access permission. By assembling corresponding (dev_i, sk_{dev_i}) in DKL, the trusted broker can obtain a key set that can be accessed by the sp_j. This process incurs minimum computation and storage cost. On the other hand, when revoking the sp_j, the trusted broker just needs to delete corresponding entries in those lists. Benefiting from the implemented "heartbeat" mechanism, no other time-consuming computations, such as device key re-issuing and data storage re-encryption are required. Therefore, SRM supports an end user to efficiently perform

access control over monitoring services and further allows to subscribe as many services as the user needs in practice.

4) Heartbeat Cost: The "heartbeat" mechanism in SRM consists of three critical functions, i.e., the *hb_loop()* at the trusted broker's side, the *hb_event()* and *assert()* at the enclave's side. By following numerical analysis, we show that the performance costs of these functions are relatively very small. The main overhead of *hb_loop()* is to encrypt the heartbeat message with the shared communication key sk_{comm}, the complexity of which depends on the underlying message size. By our design, a heartbeat only includes a 4 bytes counter c and a 1 byte revocation indicator *is_revoked*, so the encryption operation incurs very small overhead. Correspondingly, the *hb_event()* mainly performs a costless decryption operation. Lastly, the *assert()* obviously comprises no time-consuming computations.

The potential system resource cost introduced by the "heartbeat" mechanism is that both the trusted broker and monitoring service must maintain at least one dedicated thread (a trusted broker may create one thread for each monitoring service, and vise verse.) to constantly emit or handle heartbeats during their lifetime. Besides, it also consumes network resources for data transmission. Typically, the size of one heartbeat packet is 13 bytes, including 8 bytes header data and 5 bytes payload data. Given that the heartbeat frequency is hb_freq, then the total data load carried by the network in a unit time is $13 \cdot hb\_freq$ bytes.

5) Trade-Off Between System Robustness and Service Revocation Timeliness: As discussed in Section IV-D, a bigger value of $threshold$ for freshness check algorithm $assert()$ may allow a monitoring service to continue processing user data for a longer period of time in the situation where the *revocation heartbeat* is not delivered to the enclave due to either malicious OS or compromised monitoring application. In other words, the revocation timeliness cannot be guaranteed. On the other hand, if the value of $threshold$ is set as the minimum, i.e., equals to the heartbeat frequency hb_freq, the above issue is well solved, however, it may make the system suffer from frequent data processing failure when assuming a practical connectionless network environment, in which abrupt packet loss happens. Thus, the system is not robust under such situation.

Next, we try to find a balance point between the two properties. In view of user privacy, the system should always minimize the amount of potential information leak, i.e., minimizing $threshold$. Based on such simple principle, we only need to figure out what is the time interval between two adjacent received heartbeats in the worst case. Note that the two adjacent received heartbeats may not be continuously generated by the trusted broker due to packet loss. According to [22] and [23] , packet loss on Internet is highly bursty, which means that losses prefer to happen consecutively. Thus, given that the heartbeat frequency is hb_freq and the loss burst length, i.e., the number of consecutive packets lost in a burst, is l ($l \geq 0$), we can use the following equation to formulate the time interval t of two adjacent received heartbeats,

$$t = \frac{l+1}{hb\_freq}.$$

It is found that the distribution of loss burst length l can be modeled with Pareto distribution. The cumulative distribution function (CDF) is,

$$F_L(l) = 1 - l^{-\alpha} \ (\alpha > 0).$$

Thus, the time interval t also accords with Pareto distribution, with the following CDF,

$$F_T(t) = 1 - (hb\_freq \cdot t - 1)^{-\alpha} \ (\alpha > 0).$$

Then, the complementary CDF of t is,

$$\overline{F}_T(t) = P(T \geq t) = (hb\_freq \cdot t - 1)^{-\alpha} \ (\alpha > 0).$$

Suppose that $P(T \geq t) \leq \epsilon$ $(0 < \epsilon < 1)$ is negligible, then the time interval of two adjacent received heartbeats in the worst case is,

$$t_{worst} = \frac{1}{hb\_freq} \cdot (\epsilon^{-\frac{1}{\alpha}} + 1) \ (0 < \epsilon < 1, \alpha > 0).$$

As a conclusion, the relationship between the $threshold$ and the hb_freq is as follows.

$$threshold = \frac{1}{hb\_freq} \cdot (\epsilon^{-\frac{1}{\alpha}} + 1) \ (0 < \epsilon < 1, \alpha > 0),$$

where α is a Pareto parameter fit to the empirical distribution of loss burst length based on historical statistics, and ϵ is chosen carefully according to the preference of the system. Concretely, [22] computes $\alpha = 1.38$ based on its collected historical Internet packet datasets. Provided that the system sets $\epsilon = 0.001$ and $hb\_freq = 5$ (packets per second), then the $threshold$ should be approximately 30 seconds.

VII. Related Work

Attribute-Based Encryption (ABE), first proposed by Sahai and Waters [24], is a promising privacy-preserving data access control technology that achieves fine-grained access control, scalable key management and flexible data distribution. It has been well studied and adopted in many cloud computing applications in the past decade [2]–[5], [25], [26]. Recently, Wang *et al.* [27] give a comprehensive performance evaluation of ABE, focusing on execution time, data and network overhead, energy consumption, and CPU and memory usage, to understand at what cost ABE offers its benefits and under what situations ABE is best suited for use in the IoT. They concluded that the computation cost in encryption and decryption phase may be a heavy burden for those resource-limited devices. Many researchers try to leverage other powerful entities to offload the cumbersome computation. For example, Yang *et al.* [28] exploit the cloud as an outsourcing entity to encrypt data for publishers and decrypt data for receivers. Huang *et al.* [29] and Zhang *et al.* [30] delegate the computation of encryption and decryption to fog nodes, which is a micro data-center adjacent to the end user in fog computing paradigm. Our work, however, avoids such cumbersome cryptography-based methods by utilizing the light-weight hardware, i.e., Intel SGX, to achieve fine-grained access control over user's data while achieving the same security requirements in the challenging IoT scenario.

Intel SGX is a hardware-based trusted computing technology, which has been studied a lot in the literature.

Baumann *et al.* [31] implemented a prototype named Haven to protect unmodified legacy against malicious OS by running them in SGX enclaves. Arnautov *et al.* [32] and Shinde *et al.* [33] built a secure Linux container with Intel SGX to defend against outside attacks. Fisch *et al.* [15] propose a system called IRON with Intel SGX to make functional encryption (FE) and multi-input functional encryption (MIFE) practical. Sun *et al.* [34] exploit Intel SGX to address the challenging searchable encryption (SE) problem. In comparison to existing works, we solve the non-trivial key revocation issue faced by Intel SGX by introducing a "heartbeat" protocol.

Our preliminary work [21] focused on the application-level protocol flow of the proposed framework. In contrast, this paper re-defines the framework and formally describes it (Section IV). In particular, we define a new protocol called HBSetup to separately represent the initialization of the "heartbeat" mechanism, and we also define relevant function modules in each protocol, with clear *inputs* and *outputs*. Correspondingly, we refine the security properties that the framework achieves (Section V). In addition, we improve the prototype by implementing network communication stacks on both the trusted broker and monitoring service sides instead of simulation, and thoroughly evaluate the framework with additional experiments and numerical analysis (Section VI). Notably, we answer a question with regard to the parameter setting in the "heartbeat" mechanism.

VIII. Conclusion

In this paper, we propose a secure and efficient remote monitoring framework named SRM in the context of IoT, which enables two fundamental security functionalities for users, i.e, a user can control which deployed devices can be accessed by which monitoring services, and he/she can be further assured that functions over his/her data are securely executed without leaking the privacy information to unauthorized entities. To this end, we leverage the off-the-shelf secure hardware, i.e., Intel SGX to circumvent those cumbersome crypto-based solutions in previous works. Furthermore, we also introduce a "heartbeat" mechanism to solve the key revocation issue and thus efficiently support service un-subscription for users. Lastly, by implementation and evaluation, we demonstrate that SRM is feasible in practice and almost raises no performance degradation.

References

[1] M. Hassanalieragh, A. Page, T. Soyata, G. Sharma, and M. Aktas, "Health monitoring and management using Internet-of-Things (IoT) sensing with cloud-based processing: Opportunities and challenges," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jul. 2015, pp. 285–292. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7207365

[2] M. Li, S. Yu, Y. Zheng, K. Ren, and W. Lou, "Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 131–143, Jan. 2013. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6171175

[3] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9. [Online]. Available: https://www.cnsr.ictas.vt.edu/publication/INFOCOM10-Yu.pdf

[4] W. Sun, S. Yu, W. Lou, Y. T. Hou, and H. Li, "Protecting your right: Attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr./May 2014, pp. 226–234. [Online]. Available: https://www.cnsr.ictas.vt.edu/publication/Sun-TPDS-2015.pdf

[5] Z. Wan, J. Liu, and R. H. Deng, "HASBE: A hierarchical attribute-based solution for flexible and scalable access control in cloud computing," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 2, pp. 743–754, Apr. 2012. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6046132

[6] A. C. Yao, "Protocols for secure computations," in *Proc. 23rd Annu. IEEE Symp. Found. Comput. Sci.*, Chicago, IL, USA, Nov. 1982, pp. 160–164. [Online]. Available: https://research.cs.wisc.edu/areas/sec/yao1982-ocr.pdf

[7] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput.*, New York, NY, USA, 2009, pp. 169–178, doi: 10.1145/1536414.1536440.

[8] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 636–654. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7546527

[9] M. Dworkin, "Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC," NIST, Gaithersburg, MD, USA, Tech. Rep. 800-38D, 2007. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf

[10] F. McKeen *et al.*, "Innovative instructions and software model for isolated execution," in *Proc. 2nd Int. Workshop Hardw. Architectural Support Secur. Privacy*, New York, NY, USA, 2013, pp. 10:1–10:1, doi: 10.1145/2487726.2488368.

[11] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. (Jun. 2013). *Innovative Technology for CPU Based Attestation and Sealing*. [Online]. Available: https://software.intel.com/en-us/articles/innovative-technology-for-cpu-based-attestation-and-sealing

[12] E. Brickell and J. Li, "Enhanced privacy id from bilinear pairing for hardware authentication and attestation," in *Proc. IEEE 2nd Int. Conf. Social Comput.*, Aug. 2010, pp. 768–775. [Online]. Available: https://eprint.iacr.org/2009/095.pdf

[13] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen, "Intel software guard extensions: EPID provisioning and attestation services," Intel, Santa Clara, CA, USA, Tech. Rep. 2016/ww10, 2016. [Online]. Available: https://software.intel.com/en-us/download/intel-sgx-intel-epid-provisioning-and-attestation-services

[14] M. Barbosa, B. Portela, G. Scerri, and B. Warinschi, "Foundations of hardware-based attested computation and application to SGX," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Mar. 2016, pp. 245–260. [Online]. Available: https://eprint.iacr.org/2016/014.pdf

[15] B. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov, "Iron: Functional encryption using Intel SGX," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, 2017, pp. 765–782, doi: 10.1145/3133956.3134106.

[16] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado. (2016). "Inferring fine-grained control flow inside SGX enclaves with branch shadowing." [Online]. Available: https://arxiv.org/abs/1611.06952

[17] W. Wang *et al.*, "Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, 2017, pp. 2421–2434, doi: 10.1145/3133956.3134038.

[18] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on Intel SGX," in *Proc. 10th Eur. Workshop Syst. Secur.*, New York, NY, USA, 2017, pp. 2:1–2:6, doi: 10.1145/3065913.3065915.

[19] V. Costan and S. Devadas, "Intel SGX explained," Cryptol. ePrint Arch., Mihir Bellare Bennet UCSD, La Jolla, CA, USA, Tech. Rep. 2016/086, 2016. [Online]. Available: https://eprint.iacr.org/2016/086

[20] Intel Corporation, "Intel software guard extensions programming reference," Intel, Santa Clara, CA, USA, White Paper 2014/329298-002US, 2014. [Online]. Available: https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf

[21] Y. Chen, W. Sun, N. Zhang, Q. Zheng, W. Lou, and Y. T. Hou, "A secure remote monitoring framework supporting efficient fine-grained access control and data processing in IoT," in *Proc. SecureComm*, Singapore, Aug. 2018, pp. 1–20. [Online]. Available: https://www.cnsr.ictas.vt.edu/publication/yax1.pdf

[22] M. S. Borella, D. Swider, S. Uludag, and G. B. Brewster, "Internet packet loss: Measurement and implications for end-to-end QoS," in *Proc. ICPP Workshops*, Aug. 1998, pp. 3–12. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=721868

[23] S. K. Chin and R. Braun, "A survey of UDP packet loss characteristics," in *Proc. Conf. Rec. 35th Asilomar Conf. Signals, Syst. Comput.*, vol. 1, Nov. 2001, pp. 200–204. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=986905
[24] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Proc. 24th Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer-Verlag, 2005, pp. 457–473. [Online]. Available: https://eprint.iacr.org/2004/086.pdf
[25] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, New York, NY, USA, 2006, pp. 89–98, doi: 10.1145/1180405.1180418.
[26] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2007, pp. 321–334. [Online]. Available: https://www.cs.utexas.edu/~bwaters/publications/papers/cp-abe.pdf
[27] X. Wang, J. Zhang, E. M. Schooler, and M. Ion, "Performance evaluation of attribute-based encryption: Toward data privacy in the IoT," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 725–730. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6883405
[28] L. Yang, A. Humayed, and F. Li, "A multi-cloud based privacy-preserving data publishing scheme for the Internet of Things," in *Proc. 32nd Annu. Conf. Comput. Secur. Appl.*, New York, NY, USA, 2016, pp. 30–39, doi: 10.1145/2991079.2991127.
[29] Q. Huang, Y. Yang, and L. Wang, "Secure data access control with ciphertext update and computation outsourcing in fog computing for Internet of Things," *IEEE Access*, vol. 5, pp. 12941–12950, 2017. [Online]. Available: https://fardapaper.ir/mohavaha/uploads/2017/10/Secure-Data-Access-Control-with-Ciphertext.pdf
[30] P. Zhang, Z. Chen, J. K. Liu, K. Liang, and H. Liu, "An efficient access control scheme with outsourcing capability and attribute update for fog computing," *Future Generat. Comput. Syst.*, vol. 78, pp. 753–762, Jan. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X16307567
[31] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," *ACM Trans. Comput. Syst.*, vol. 33, no. 3, pp. 8:1–8:26, Aug. 2015, doi: 10.1145/2799647.
[32] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Conf. Operating Syst. Design Implement.* Berkeley, CA, USA: USENIX Association, 2016, pp. 265–283. [Online]. Available: http://dl.acm.org/citation.cfm?id=3026877.3026899
[33] S. Shinde, D. L. Tien, S. Tople, and P. Saxena, "Panoply: Low-TCB linux applications with SGX enclaves,"in *Proc. NDSS*, 2017, pp. 1–15. [Online]. Available: https://www.comp.nus.edu.sg/~tsunami/papers/panoply_ndss17.pdf
[34] W. Sun, R. Zhang, W. Lou, and Y. Thomas Hou, "Rearguard: Secure keyword search using trusted hardware," in *Proc. IEEE INFOCOM*, Mar. 2018, pp. 1–9. [Online]. Available: https://www.cnsr.ictas.vt.edu/publication/Seks.pdf

Yaxing Chen received the B.S. degree in software engineering from Northwestern Polytechnical University in 2012. He is currently pursuing the Ph.D. degree with the Department of Computer Science, Xi'an Jiaotong University. He was a Visiting Student with Virginia Tech from 2016 to 2018. His research interests lie in data security and privacy, cloud computing, with a focus on data access control and trusted computing.

Wenhai Sun received the Ph.D. degree in computer science from Virginia Tech and the Ph.D. degree in cryptography from Xidian University. He is currently an Assistant Professor with the Department of Computer and Information Technology, Purdue University, where he is also a Faculty Member of the Center for Education and Research in Information Assurance and Security. His studies cover various topics in cybersecurity research, with a focus on security and privacy issues in networked information systems and cyber-physical systems, including cloud computing, distributed ledger networking, Internet of Things, computer/mobile systems, and wireless networking. He received the Distinguished Paper Award from the ACM ASIACCS in 2013.

Ning Zhang received the Ph.D. degree from Virginia Tech in 2016. He is currently an Assistant Professor with the Department of Computer Science and Engineering, Washington University in St. Louis. Before that, he was with an industry as a cyber engineer and a technical lead for over ten years. His research focus is system security, which lies at the intersection of security, embedded systems, and computer architecture and software.

Qinghua Zheng received the Ph.D. degree in system engineering from Xi'an Jiaotong University. He is currently the Leader of the Innovation Team, National Natural Science Foundation of China, the Innovative Team, Ministry of Education, and the Shaanxi Key Scientific and Technological Innovation Team. He was a recipient of the National Funds for Distinguished Young Scientists and a Distinguished Professor for the Changjiang River Scholar Project in China. He is with the first batch of leading scientists of the Ten-Thousand Talents Project and also the candidate of the New Century National Hundred Thousand- and Ten Thousand-Talents Project in China. His major research fields include intelligent e-learning, big data mining and applications, and software reliability.

Wenjing Lou (F'15) received the Ph.D. degree in electrical and computer engineering from the University of Florida in 2003. From 2003 to 2011, she was a Faculty Member with the Worcester Polytechnic Institute. She has been a Professor with Virginia Tech since 2011. From 2014 to 2017, she was the Program Director of the U.S. National Science Foundation, where she was involved in the Networking Technology and Systems Program and the Secure and Trustworthy Cyberspace Program. Her current research interests focus on privacy protection techniques in networked information systems and cross-layer security enhancement in wireless networks by exploiting intrinsic wireless networking and communication properties.

Y. Thomas Hou (F'14) received the Ph.D. degree from the NYU Tandon School of Engineering (formerly Polytechnic University) in 1998. He joined Virginia Tech, Blacksburg, VA, USA, in 2002, where he is currently a Bradley Distinguished Professor of Electrical and Computer Engineering. From 1997 to 2002, he was a Member of Research Staff with Fujitsu Laboratories of America, Sunnyvale, CA, USA. His current research focuses on developing innovative solutions to complex science and engineering problems arising from wireless and mobile networks. He is also interested in wireless security. He has over 250 papers published in the IEEE/ACM journals and conferences. His papers were recognized by five best paper awards from the IEEE and two paper awards from the ACM. He holds five U.S. patents. He has authored/co-authored two graduate textbooks: *Applied Optimization Methods for Wireless Networks* (Cambridge University Press, in 2014) and *Cognitive Radio Communications and Networks: Principles and Practices* (Academic Press/Elsevier, in 2009). He was named an IEEE Fellow for contributions to modeling and optimization of wireless networks. He was/is on the editorial boards of a number of IEEE and ACM transactions and journals. He is the Steering Committee Chair of the IEEE INFOCOM Conference and a member of the IEEE Communications Society Board of Governors. He is also a Distinguished Lecturer of the IEEE Communications Society.