A Secure Remote Monitoring Framework Supporting Efficient Fine-grained Access Control and Data Processing in IoT

Yaxing Chen[1,2], Wenhai Sun[2], Ning Zhang[2], Qinghua Zheng[1], Wenjing Lou[2], and Y. Thomas Hou[2]

[1] School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an Shaanxi 710049, China
cyx.xjtu@gmail.com, qhzheng@mail.xjtu.edu.cn

[2] Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg VA 24060, USA
{whsun, ningzh, wjlou, thou}@vt.edu

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 (RPM) 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 provider (HCP) for processing and decision making. As the measurements involve private patient information, access control, confidentiality, and trustworthy processing of the 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 patient and HCP sides. In this work, we deviate from the conventional software-based solutions and propose a secure and efficient remote monitoring framework using latest hardware-based trustworthy computing technology, such as Intel SGX. In addition, we present a robust and lightweight "heartbeat" protocol to handle notoriously difficulty user revocation problem. We implement a prototype of the framework for PRM and show that the proposed framework can protect user data privacy against unauthorized parties, with minimum performance cost compared to existing software-based solutions with such strong privacy protection.

Keywords: Remote patient monitoring · Internet-of-Things (IoT) · Fine-grained access control · Secure hardware · Trusted computing.

1 Introduction

Remote patient monitoring is one of the silver applications of the Internet of Things (IoT) system. It allows health care providers 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 proprietary 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 PRM system. The patient can permit an authorized HCP to access data types based on the offered service. For instance, a cardiovascular HCP 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 date 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. RPM at the client side consists of a number of battery-powered and extremely resource-constrained devices, which are likely unable to afford complex computationally-intensive cryptographic operations. Another challenge is the realization of on-demand user 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 take the RPM as a case study and propose a secure and efficient remote monitoring framework. In contrast to the software-based solutions that exploit cryptographic primitives as building blocks, we present a novel framework by leveraging the hardware-based trusted computing technology, such as Intel SGX to protect user data privacy and enable secure computations over sensitive data. Specifically, 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. Due to the use of secure hardware, our scheme is very efficient as we only adopt symmetric encryption, such as AES and carry out the monitoring service (i.e., HCP) functions which could be arbitrary constitution 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 framework. By our design, the secret keys on the untrusted cloud server never leave the enclave (SGX term for TEE) and the trustworthy executions of the access control enforcement and HCP application are guaranteed by SGX functions. However, strong attackers, such as OS and VM hypervisor, can still launch denial-of-service (DoS) attack [9] to compromise the system. For example, it is expected that the trusted broker can explicitly inform the HCP enclaves to erase the corresponding secret keys to revoke the access permission of the HCP. However, a malicious OS may ignore such request and help the revoked HCP to continue reading the patient's data. Worse still, an HCP may be compromised and fail to invoke corresponding enclave functions in response to the revocation request. In order to solve this problem, we propose a "heartbeat" protocol. In a nutshell, we force the enclave of the revoked HCP to be unavailable if it does not receive a valid heartbeat signal from the trusted broker after the defined time window.

Our contribution can be summarized as follows.

- Building upon recent development of secure processor, we propose a practical secure remote monitoring framework that 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 less performance overhead.
- We propose a novel "heartbeat" protocol to address the drawback 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 implemented a prototype of the framework for remote patient monitoring. Experiments show that the proposed system offers unique protection with little performance overhead. The software has been open-sourced for the community to build upon the existing work.

The rest of this paper is organized as follows. Section 2 introduces the technique of Intel SGX. Section 3 gives a description of the system model, threat model, and design goals. We present the details of our framework and "heartbeat" protocol in Section 4, and analyze its security properties in Section 5. We describe the implementation of our prototype in Section 6 and evaluate it in terms of performance and framework scalability. Section 7 reviews the literature related to our work. Finally, we conclude in Section 8.

2 Background

In this section, we provide background knowledge about the used trusted hardware primitive – Intel software guard extensions (SGX).

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[3], 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. Next, we will provide some technical details of these two functions, which are essential building blocks of our framework.

Storage Sealing Intel SGX platform maintains a seal key to enable cryptographic sealing function, which is derived from a base key called Root Seal Key that is hardcoded when the Intel SGX enabled processor is manufactured. The derivation algorithm supports two policies for data accessibility control. One policy named *sealing to the enclave's identity* bases the seal key on the value of the enclave's MRENCLAVE, which is a SHA-256 digest of an internal log that records all activities done while the enclave is built. It enforces that only the certain enclave can recover sealed data. The other policy is called *sealing to the sealing identity*, which utilizes the value of the enclave's MRSIGNER to generate the seal key. The MRSIGNER is a hash of the public key of the party who signs the enclave prior to distribution. Such a policy facilitates the scenario where an enclave needs to share its sealed data with other enclaves signed by the same party.

Remote Attestation To enable this functionality, 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, i.e., MRENCLAVE and MRSIGNER, some meta-data and a MAC. The MAC is calculated using a report key derived from the Root Seal Key. After the REPORT is received, the quoting enclave then verifies it by re-computing the MAC over the underlying data of the REPORT with the same report key. 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. In other words, the firmware and hardware of the challenged enclave are trustworthy. Next, the quoting enclave generates a new signed structure called QUOTE by re-signing the underlying data of the report with the Intel Enhanced Privacy ID (EPID), which is an anonymous group signature scheme implemented by Intel. Finally, the QUOTE is delivered to the entity who in turn transfers it to the Intel Attestation Service (IAS) for validation. In principle, any verifiers that possess the group public key can verify the QUOTE.

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

Intel SGX, however, is known to be vulnerable to various physical and software attacks, for example, side-channel attacks [12, 13] including cache-timing attack, power analysis attack, branch shadowing attack, etc. Further, the compromised OS can launch DoS attack to disrupt the enclave function as it is still in charge of the underlying resource allocation. Thus, this attack on an intuitive SGX-based RPM system allows the HCP to continue accessing the patient data by dropping off the HCP revocation command from the patient.

3 Problem Formulation

3.1 System Model

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

3.2 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 HCP revocation under the DoS attack by the compromised OS or the malicious host HCP application.

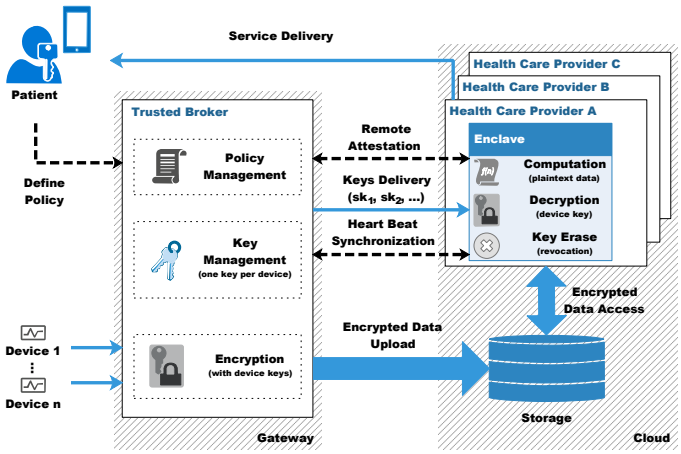

Fig. 1. The proposed framework for remote heath monitoring and protocol flows

3.3 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 patient to subscribe as many HCPs 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 personal health data and keys*:** It is expected that the measured patient data are well protected when stored and processed in the cloud and the corresponding secret keys will not be disclosed to unintended parties.
- ***Trusted HCP data processing*:** The data processing operations of HCP in the cloud should be verifiable and comply with the prescribed service agreement.
- ***Fine-grained data access control*:** An authorized HCP can only access the data types defined by the patient.
- ***Robust revocation*:** The patient should be able to revoke existing HCPs in the case of service unsubscription.

4 Our Proposed Framework

4.1 Main Idea

In our proposed framework, it is expected that the private patient data should be securely processed and also compliant with the subscribed HCP service. In

order to achieve this, we leverage Intel SGX to create an enclave for the patient 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 HCP functions.

In addition, the patient should be able to enforce access control policy for the subscribed HCPs over his/her outsourced data and revoke the access permission of the unsubscribed HCP. 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 patient can also control which HCP can access what types of patient data by providing the corresponding secret keys. Intuitively, in order to revoke an existing HCP and prevent it from further accessing the patient data, we may re-encrypt the data type that was allowed for the target HCP with updated device keys and redistribute these keys to the remaining affected HCPs. Obviously, this method incurs considerable computation and communication overhead, and cannot revoke the access permission promptly, which is very important for a real-time RPM system. The patient can also choose to explicitly send a revocation command to the enclave to destroy all assigned device keys, but it will fail if the compromised OS or HCP host application intercepts this request. To solve this challenging issue, we present a "heartbeat" protocol in our framework. The core idea is to send a periodical heartbeat signal from the patient side to retain the HCP 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 will be no longer available.

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

4.2 Framework Description

The proposed scheme comprises five steps: *System Setup*, *Data Upload*, *Service Subscription*, *Secure Data Processing*, *Service Unsubscription*. Next, we describe them in details. The main notations are summarized in Table 1.

System Setup In this phase, the patient first bootstraps and configures the trusted broker in the gateway. In particular, an access control list ACL, a device key list DKL and a secret shared key list $SSKL$ are initialized. Then the patient registers all monitoring devices to the trusted broker, who invokes the key management function $\mathbb{F}_{KM}$ to generate a unique secret key sk_i for each registered device i. The key along with the corresponding device ID i is recorded in DKL. On the other side, an HCP sets up its service application in the cloud.

Data Upload The monitoring devices constantly collect data from the patient and ambient environment, and send them as files to the trusted broker.

Table 1. Main Notations

Notation	Description
$\mathbb{E}_{App}$	The enclave launched by the health care provider.
$\mathbb{F}_*$	The function implemented by the trusted broker. $*$ can be KM for key management, Enc for encryption.
ssk_p	the shared key generated by the trusted broker for secure communication with the health care provider p.
sk_i	The secret key for device i.
ct_i^j	The ciphertext of the data file j bound to device i encrypted with sk_i
CT_i	The ciphertexts bound to device i.
ζ_p	The access rule defined for the health care provider p.
ACL	The access control list maintained by the trusted broker.
DKL	The device key list maintained by the trusted broker.
$SSKL$	The secret shared key list maintained by the trusted broker.

Each data file j from a particular device is further encrypted into the ciphertext ct_i^j by the encryption function $\mathbb{F}_{Enc}$ using the corresponding device key sk_i. Finally, the ciphertexts are uploaded to the cloud storage and organized as per device, i.e., $CT_i = \{ct_i^j | j \in F_i\}$, where F_i represents the whole file set of device i.

Service Subscription When the patient subscribes to an HCP p, he/she first defines the access permission rule ζ_p in accordance with the service agreement. ζ_p explicitly indicates which monitoring devices can be accessed by the HCP. Then, the HCP ID p along with the access rule ζ_p is recorded in the ACL. Meanwhile, the cloud HCP application initializes a dedicated enclave $\mathbb{E}_{App}$ for the target patient. Next, the trusted broker on behalf of the patient begins the remote attestation interaction with the HCP enclave $\mathbb{E}_{App}$ to ensure that all the enclave functions comply with the service agreement. At the end of a successful attestation, a secret key ssk_p is negotiated and shared between the trusted broker and the HCP enclave to generate an authenticated secure channel for subsequent communications. The trusted broker adds (ssk_p, p) to the $SSKL$ and the HCP saves the ssk_p with an internal variable *shared_key*.

Secure Data Processing Initially, the application enclave $\mathbb{E}_{App}$ of HCP p needs first to request corresponding device secret keys from the trusted broker.

After receiving the key request, the trusted broker sends back the device keys according to the defined access rule in ACL. The communication channel is protected using the shared secret key ssk_p. Next, so long as the $\mathbb{E}_{App}$ is not closed by the host application, it can constantly load the intended ciphertexts of patient data from the cloud storage, decrypt them with the obtained device keys and process the plaintext information inside the enclave. In case the enclave is torn down either due to power event or by the application itself, the secret materials can be sealed to the untrusted storage for long-term service delivery. Notably, we limit the enclave to use *sealing to the enclave's identity* policy for storage sealing, so that the obtained secret keys won't be shared with other enclaves not verified by the patient.

Service Unsubscription The patient is able to unsubscribe a particular HCP service by revoking all the assigned device keys. We propose a lightweight "heartbeat" protocol to enable efficient and robust HCP revocation. In general, the trusted broker adopts an auxiliary function, which will periodically send a heartbeat signal to the HCP enclave $\mathbb{E}_{App}$ after giving out the device keys. The signal carries a state indicating whether or not the HCP has been revoked. Upon receiving a heartbeat signal with revocation state, the $\mathbb{E}_{App}$ will erase all secret keys. Otherwise, it updates an internal variable named *hb_state*, which is critical to sustaining the functionality of enclave. If *hb_state* is not updated after a defined time window, all the functions of the enclave towards secure data processing cannot be executed properly. Thus, it can prevent further data access by the HCP. In what follows, we describe the "heartbeat" protocol in details.

4.3 Heartbeat Protocol

Fig. 2 shows the proposed "heartbeat" protocol, which runs between the trusted broker and the HCP.

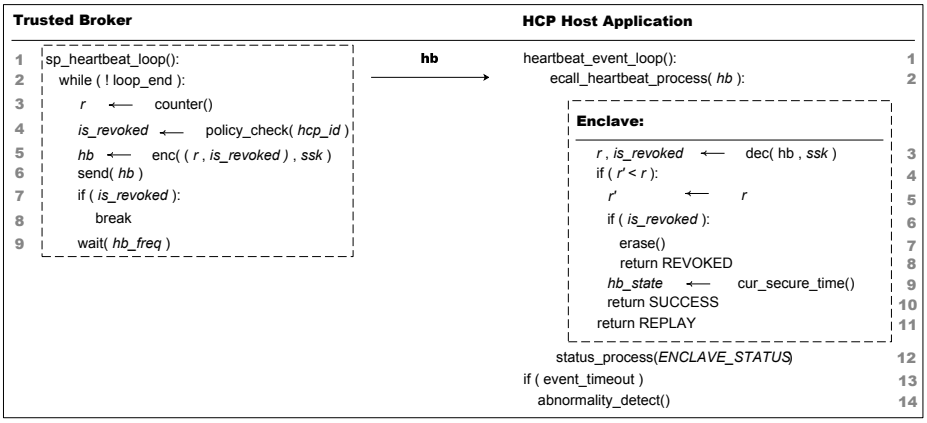

Fig. 2. The "heartbeat" protocol between the trusted broker and HCP enclave

On the trusted broker side. A loop function is implemented to enable periodical heartbeat signal emission and each iteration represents a heartbeat cycle. It also uses a variable *loop_end*, initialized as *false*, to control the on-off switch of emitting heartbeat signal (line 2). During each heartbeat cycle, the trusted broker first calls the *counter()* function to obtain a monotonically increased positive number r (line 3). Then it calls the *policy_access()* function to get the current revocation status of the target HCP, which is stored in a boolean variable *is_revoked* (line 4). It generates a heartbeat signal *hb* by using authenticated encryption, such as GCM[AES] to encrypt r and *is_revoked* with the shared key *ssk* of the target HCP and sends it to the HCP host application in the cloud (line 5 and 6). If the HCP has not been revoked, the current process will be suspended for a defined period *hb_freq* before entering the next cycle. Otherwise, it will exit the loop and stops sending the heartbeat signal (line 7-9).

On the HCP host application side. The HCP implements an event response function named *heartbeat_event_loop()* to monitor heartbeat signals (line 1), in which it transfers the received heartbeat message to its enclave by calling the enclave function *ecall_heartbeat_process()* (line 2). Within the enclave, it decrypts the message with the shared key *ssk* to recover the number r and the revocation status *is_revoked* (line 3). Next, it checks whether r is larger than the number *r'*, which is maintained by the enclave to record the maximum of r that has been received before (line 4). Note that *r'* is initialized to be -1. If $r \leq r'$, the enclave returns the state of REPLAY (line 11). Otherwise, the enclave stores r as new *r'* (line 5). Then it checks the revocation status (line 6). Provided that the HCP needs to be revoked, the *erase()* function will be called to free the memory for storing the obtained secret keys from the trusted broker, and return the state of REVOKED to host application (line 7 and 8). Otherwise, the enclave updates a global variable *hb_state* by invoking the *cur_secure_time()*, which returns a trusted machine time. The enclave also returns the state of SUCCESS to the host application (line 9 and 10).

The host application bases the returned status to do some post processing (line 12), i.e., REPLAY, REVOKED and SUCCESS. Specifically, REPLAY indicates that the enclave suffers from the replay attack. REVOKED represents that the HCP has been revoked. SUCCESS means that the current event is successfully processed. In addition, if the HCP doesn't receive a heartbeat message from the trusted broker for a defined time period *event_timeout* and the REVOKED status has not yet been set, it may suffer from abnormalities, either network failure or DoS attack, and thus triggers the detection function (line 12, 13), which is out of the scope of this paper.

To further enable the revocation mechanism, we need to enhance other enclave functions by inserting an assert before normal function codes are executed, which checks the freshness of the *hb_state*. Fig. 3 shows the checking algorithm. First, it gets the current trusted machine time *cur_time* by invoking *cur_secure_time()* and computes the difference *diff_time* between *cur_time* and *hb_state* (line 2,3). If *diff_time* is less equal than a defined time window named *threshold*, then it returns true, meaning that the *hb_state* is fresh and that the

subsequent codes can be properly executed (line 4,5). Otherwise, it returns false (line 6,7). 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. Supposed that the *threshold* were very large compared to the heartbeat frequency *hb_freq*, the mechanism can be robust to temporary network failure or compromised OS. However, it will postpone the revocation time of taking effect. For example, when the revocation heartbeat signal is not received by the HCP enclave because of network failure, the HCP can continue processing the patient's data until the time window defined by *threshold* runs out. On the contrary, if the *threshold* is close to the *hb_freq*, it can response revocation event in time but may be vulnerable to the network failure and compromised OS.

```
Enclave:
1 assert():
2     cur_time  ←—   cur_secure_time()
3     diff_time ←—   cur_time - hb_state
4     if (diff_time <= threshold):
5         return true
6     else:
7         return false
```

Fig. 3. The freshness check assert

Remark The proposed "heartbeat" protocol can achieve the desired HCP revocation function. Its correctness can be guaranteed by the follows. In the case of receiving the valid heartbeat signal in the defined time window, if *is_revoked* is false, the HCP can continue to access the patient data. Otherwise, the access permission of the HCP will be revoked by erasing all the assigned secret keys in the enclave. Should the heartbeat signal is not received by the enclave during the defined time window, the abnormality, due to either network delay or the intentional drop off of the revocation signal by the compromised OS or HCP host application, will be detected, which disables the remaining critical HCP enclave functions towards data processing.

5 Security Analysis

In this section, we show that our proposed scheme can achieve the defined security goals.

5.1 Confidentiality of Personal Health Data and Provisioned Key

This property is satisfied by both software-based encryption algorithms, such as AES, and the used secure hardware TEE function, i.e. Intel SGX enclave.

When outside the enclave, the patient information collected from various monitoring devices are encrypted using respective device keys and stored in the cloud. After remote attestation, the relevant device keys are provisioned into enclave through an authenticated secure channel. The encrypted patient data can only be decrypted and processed inside the enclave. On the other hand, the shared secret key and assigned device keys by our design never leave the enclave. Thus, the confidentiality of the data and relevant keys are realized in this work.

5.2 Trusted HCP Data Processing

This property is guaranteed by the remote attestation function of Intel SGX. During this process, the patient will verify the integrity and correctness of critical HCP functions that take his/her private data as input. Thus, the patient can be assured of the trustworthy execution of the subsequent data processing and its compliance with the subscribed service agreement.

5.3 Fine-grained Data Access Control

We use different device-wise keys to encrypt each data type associated with this device. Thus, the patient 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 HCP. This is enforced by only giving the HCP the relevant secret device keys.

5.4 Robust HCP Revocation

We leverage the "heartbeat" protocol to efficiently and effectively revoke an existing HCP from the system. The correctness has been stated in section 4.3. Here we focus on the other two security-related aspects.

- ***Non-forgeability*:** No other parties except for the trusted broker and HCP enclave can access the shared secret key, which is used to encrypt and authenticate the heartbeat messages.
- ***Replay attack resistance*:** A compromised party, e.g. OS, HCP host application, may replay previously received heartbeat message to the enclave to keep the freshness of *hb_state*. However, we use a monotonically increased number r to maintain the message order. It is expected that r in newly received heartbeat message should be greater than the stored r' in the enclave. Otherwise, the replay attack can be detected.

6 Implementation and Evaluation

We implemented a prototype[4] in C using the Intel SGX SDK 2.1 for Linux, and enclaves are built as Linux Shared Objects (.so). Our prototype is tested

[4] The project is available to access through the GitHub via the following link: https://github.com/yxChen1990/SGXLAB.git

on an Intel NUC7i5BNH, an SGX enabled platform running 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. Currently, an Intel license is required to build enclaves in release mode, so we compiled the code using g++ in a debug mode.

6.1 Implementation

In our prototype, we implemented the network communication interfaces invoked by the HCP host application with directly stub function calls from the trusted broker. We also implemented a data sample module to imitate the activities of monitoring devices. In particular, it provides a stub function *data_send()* for directly invocation by the trusted broker. Besides, we omitted the cloud storage using a stub function *sp_upload_data()* implemented in the trusted broker, which encrypts data sent by the sample module and is further invoked by the HCP host application for loading the ciphertext data. Lastly, the HCP enclave provided abundant ECALL functions for the HCP host application to accomplish designed protocols. Below, we will give the description of each interface in accordance with its functionality.

Remote Attestation The functions in this module enable the trusted broker to validate the hardware and software TCB of the HCP enclave and agree on the secret shared key between the two entities. Referring to the sample code provided by Intel SDK, we implemented this mechanism by negotiating five core messages between the trusted broker and HCP host application, which are denoted by msg_0, msg_1, msg_2, msg_3, msg_{ret}, respectively. Specifically, the msg_0 carries an Extended GID generated by the HCP host application. It is processed by *sp_ra_proc_msg0_req()* in the trusted broker to validate the HCP host application before launching remote attestation. Provided that the validation was passed, the HCP host application will initialize remote attestation by invoking *ecall_init_ra()*, which returns an attestation context. Based on the attestation context, the msg_1 including the DHKE public key of the HCP enclave is constructed and sent to the trusted broker. In response, the trusted broker calls *sp_ra_proc_msg1_req()* to process msg_1 and returns back msg_2, involving the DHKE public key of the trusted broker. At the moment, a 128-bit asymmetric secret shared key between the trusted broker and HCP enclave can be constructed. Notably, if the final attestation is successful, the shared key will be recorded at both side, i.e., the trusted broker inserts it along with the HCP's ID, denoted by *(hcp_id, ssk)* to the SSKL and the HCP enclave writes it to the global variable *shared_key*. In the last round communication, the msg_3, representing the QUOTE generated for the specific HCP enclave, is sent to the trusted broker for verification. Instead of communicating with the IAS, the *sp_ra_proc_msg3_req()* locally verifies the MRENCLAVE and MRSIGNER and returns back the final attestation result msg_{ret}. On the HCP's end, it invokes *ecall_verify_att_result_mac()* to verify msg_{ret} and further does some post-processing.

Heartbeat This module includes functions used to synchronize heartbeat messages between the trusted broker and HCP enclave. Following the protocol design in Section 4.3, we implemented a *sp_heartbeat_loop()* at the trusted broker's side to constantly emit heartbeat signal msg_{hb} to the HCP enclave and an *ecall_heartbeat_process()* at the HCP enclave's side to handle the captured heartbeat event. In particular, we created a dedicated thread to simulate the protocol execution.

Key Management This function module facilitates the trusted broker to generate device keys and distribute them to HCP enclaves. The trusted broker maintains two lists, i.e., device key list (DKL) and secret shared key list (SSKL) to support such a functionality. We use two struct arrays to implement them: the first one is defined as *(dev_id, sk)*; the second one is defined as *(hcp_id, ssk)*. It also implements three functions. Specifically, the *key_generate()* function is used to generate keys for registered monitoring devices. The *key_access()* interface enables the access of the two lists by other functions in the trusted broker. To facilitate key distribution, it implements a *sp_km_proc_key_req()* function to deal with key requests from HCP enclaves, which takes the key request message msg_{req} as input and returns back msg_{sk}. More specifically, the msg_{req} includes the HCP ID *hcp_id* and its corresponding ciphertext generated by the HCP enclave using the shared key *shared_key*. The msg_{sk} is a struct encrypted with the same shared key retrieved from SSKL. The underlying struct consists of the key number and target device keys, and can only be recovered to a global variable *device_keys* within the HCP enclave through the invocation of *ecall_put_keys()* by the HCP host application.

Seal Secrets With regard to the HCP, we offer two ECALL functions in this module to enable that keys received by the HCP enclave can be flushed out to the secondary storage for long-term service provision. The *ecall_create_sealed_policy()* encrypts keys with the platform *seal key* and returns the ciphertext data to the HCP host application, which in turn can be stored in the untrusted storage medium. On the contrary, the *ecall_perform_sealed_policy()* recovers the sealed keys into the HCP enclave. By our design, the exploited policy for deriving the *seal key* is limited to only use *sealing to the enclave's identity*, such that an upgraded enclave need to once again attest to the trusted broker and request keys from it.

Policy Management For the trusted broker, we also implemented related interfaces to accomplish policy management. The *sp_define_policy()* is provisioned to facilitate a patient to define his/her access policy towards HCPs by inserting access rules to the access control list (ACL). The ACL is implemented by a struct array and the struct is defined as *(hcp_id, dev_id, dev_id, ...)*. Correspondingly, we implemented a *policy_access()* function to allow the access of ACL by other functions.

Data Processing Provided that all above modules were properly functioned, the HCP enclave then could compute over patient's encrypted data. We implemented an *ecall_perform_statistics()* function as an example, which takes

two encrypted data as inputs and outputs some statistic measurements like mean and variance of the underlying data.

Last but not the least, to support user revocation, we augment all above defined ECALL functions except those in Heartbeat and Remote Attestation modules by enforcing the freshness assert checking at the point where the function starts. In particular, the freshness time window *threshold* within the assert algorithm is set up as 5 times of the heartbeat frequency *hb_freq*.

6.2 Evaluation

As shown in **Implementation**, our framework involves many function modules. The evaluation of the system aims to answer the following questions:

- How is computation performance when using Intel SGX?
- What is the cost by introducing the heartbeat mechanism?
- How is the scalability of the proposed framework in terms of fine-grained access control?

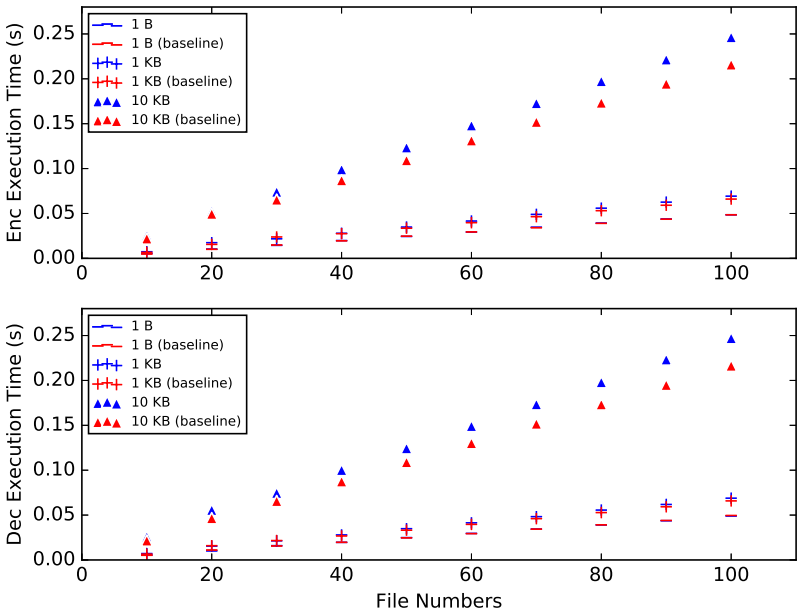

Fig. 4. The computation performance w and w/o Intel SGX

Constant time functions. Majority of the aforementioned functions are invoked few times during protocol execution and have constant overhead for each

revocation. For example, the functions in key management at both sides either perform an $O(1)$ symmetric *key generation* algorithm or conduct an $O(1)$ *read/write* operation on defined data structure. Similarly, functions in seal secrets module perform a symmetric *encrypt/decrypt* operation and functions in policy management perform a list *read/write* operation. In addition, though the remote attestation between the trusted broker and HCP enclave needs multiple network communications and complex verification computations, it is a one-time protocol finished at the service subscription phase for a given HCP. Therefore, it incurs no performance degradation to the subsequent data processing.

Performance by SGX enclave. In this experiment, we aim to measure the performance penalty when using Intel SGX. Specifically, we implement a 128bit AES-GCM scheme and 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 $4933kbps$ 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. 4 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 128bit AES-GCM function to provide the message confidentiality and integrity. It is expected that this optimized AES function will give us a much better performance compared to our own implementation. We will apply this function and evaluate its practical performance in the future.

Heartbeat cost. The heartbeat mechanism in our framework consists of three critical functions, i.e., the *sp_heartbeat_loop()* at the trusted broker's side, the *ecall_heartbeat_process()* and *freshness_assert()* at the HCP enclave's side. By following numerical analysis, we show that the performance costs of these functions are relatively very small. The main cost of *sp_heartbeat_loop()* is to encrypt the heartbeat message with the shared symmetric key, the complexity of which depends on the underlying message size. By our design, the size of heartbeat message is fixed, including a 4 bytes *counter* and a 1-byte *is_revoked*, so the performance cost can be ignored. Accordingly, in the *ecall_heartbeat_process()*, it mainly performs a reverted decryption operation. Lastly, the *freshness_assert()* obviously comprises no time-consuming operations.

The only potential resource cost introduced by the heartbeat mechanism is that both the trusted broker and HCP host application must maintain a dedicated thread to constantly emit or handle heartbeat messages during the lifetime of the service.

Scalability of the framework in terms of fine-grained access control. On behalf of the patient, a trusted broker is established in the gateway to control the access of his/her monitoring devices by multiple health care providers. In theory, our framework can support the end user to subscribe as many HCPs as he/she needs in practice. On one hand, to accomplish access control, the trusted broker only maintains numbered shared device keys in the DKL for data encryption as per device and two unique tuples for each subscribed HCP, i.e., *(hcp_id, dev_id[])* in the ACL to indicate which device keys can be accessed by the HCP enclave and *(hcp_id, ssk)* in the SSKL to secure the subsequent communication between the two entities, which incurs minimum computation and storage cost. On the other hand, by implementing the heartbeat mechanism, an HCP can be revoked without triggering other time-consuming computations, such as device key re-issuing and data storage re-encryption.

7 Related Work

Attribute-Based Encryption (ABE), first proposed by Sahai and Waters [14], 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, 15, 16]. Recently, Wang et al. [17] 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. [18] exploit the cloud as an outsourcing entity to encrypt data for publishers and decrypt data for receivers. Huang et al. [19] and Zhang et al. [20] 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. [21] implemented a prototype named Haven to protect unmodified legacy applications against malicious OS by running them in SGX enclaves. Arnautov et al. [22] and Shinde et al. [23] built a secure Linux container with Intel SGX to defend against outside attacks.

Fisch et al. [24] propose a system called IRON with Intel SGX to make functional encryption (FE) and multi-input functional encryption (MIFE) practical. Sun et al. [25] 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.

8 Conclusion

In this paper, we propose a secure and efficient framework for remote patient monitoring in the context of IoT, which enables two fundamental security functionalities for users (patients), i.e, a user can control which deployed devices can be accessed by which monitoring services (HCPs), 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 efficiently support service unsubscription for users. Lastly, by implementing a prototype, we demonstrate that our framework is feasible in practice and almost raises no performance degradation.

Acknowledgement

This work was sponsored by National Key Research and Development Program of China under Grant No. 2016YFB1000303, Innovative Research Group of the National Natural Science Foundation of China (61721002), Innovation Research Team of Ministry of Education (IRT_17R86), the National Science Foundation of China under Grant Nos. 61502379, 61532015 and 61672420, Project of China Knowledge Center for Engineering Science and Technology, and China Scholarship Council under Grant No. 201606280105. This work was also supported in part by US National Science Foundation under grants CNS-1446478 and CNS-1443889.

References

1. Hassanalieragh M., Page A., Soyata T.: Health monitoring and management using Internet-of-Things (IoT) sensing with cloud-based processing: Opportunities and challenges. In: IEEE SCC 2015, (2015)
2. Li M., Yu S., Zheng Y., Ren K., Lou W.: Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. IEEE TPDS, 24(1), 131-143 (2013).
3. Yu S., Wang C., Ren K., Lou W.: Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing. In: IEEE INFOCOM 2010, pp. 1-9 (2010)
4. Sun W., Yu S., Lou W., Hou Y. T., Li H.: Protecting your right: Attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. In: IEEE INFOCOM 2-14, pp. 226-234 (2014)

5. Wan A., Liu J., Deng R. H.: Hasbe: A hierarchical attribute-based solution for flexible and scalable access control in cloud computing. IEEE TIFS, 7(2), pp. 743-754 (2012)
6. Yao A.C.: Protocols for secure computations. In: IEEE SFCS 1982, pp. 160-164(1982)
7. Gentry C.: Fully homomorphic encryption using ideal lattices. In: ACM STOC 2009, pp. 97-105 (2009)
8. Fernandes E., Jung J., Prakash A.: Security analysis of emerging smart home applications. In: IEEE S&P 2016, pp. 636-654 (2016)
9. Costan V., Devadas S.: Intel SGX Explained. In: IACR Cryptology ePrint Archive, 86, (2016)
10. McKeen F., Alexandrovich L., Berenzon A., Rozas C., ShafiH.: Innovative instructions and software model for isolated execution. In: Hardware and Architectural Support for Security and Privacy 2013, (2013)
11. Anati I., Gueron S., Johnson S. P., Scarlata V. R.: Innovative technology for CPU based attestation and sealing. In: Hardware and Architectural Support for Security and Privacy 2013, (2013)
12. Lee S., Shih M., Gera P., Kim T., Kim H., Peinado M.: Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In: USENIX Security Symposium 2017, pp. 557-574 (2017)
13. Wang W., Chen G., Pan X., Zhang Y., Wang X., Bindschaedler V., Tang H., Gunter C. A.: Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In: ACM CCS 2017, pp. 2421-2434 (2017)
14. Sahai A. Waters B.: Fuzzy dentity-based encryption. In: EUROCRYPT 2005, pp. 457-473 (2005)
15. Goyal V., Pandey O., Sahai A., Waters B.: Attribute-based encryption for fine-grained access control of encrypted data. In: ACM CCS 2006, p. 89 (2006)
16. Bethencourt J., Sahai A., Waters B.: Ciphertext-Policy Attribute-Based Encryption: In: IEEE S&P 2007, pp. 321-334 (2007)
17. X. Wang, J. Zhang, E. M. Schooler, and M. Ion: Performance evaluation of attribute-based encryption: Toward data privacy in the IoT. In: IEEE ICC 2014, pp. 725-730 (2014)
18. Yang L., Humayed A., Li F.: A multi-cloud based privacy-preserving data publishing scheme for the Internet of Things. In: ACM ACSAC 2016, pp. 30-39 (2016)
19. Huang Q., Yang Y., Wang L.: Secure data access control with ciphertext update and computation outsourcing in fog computing for Internet of Things. IEEE Access, 5, pp. 12941-12950 (2017)
20. Zhang P., Chen Z., Liu J. K., Liang K., Liu H.: An efficient access control scheme with outsourcing capability and attribute update for fog computing. Future Generation Computer System, 78(2) , pp. 753-762 (2018)
21. Baumann A., Peinado M., Hunt G.: Shielding applications from an untrusted cloud with Haven. ACM TCS, 33(3), pp. 1-26 (2015)
22. Abadi M. Barham P., Chen J., et al: TensorFlow: A system for large-scale machine learning. In: Usenix OSDI 2016, pp. 265-284 (2016)
23. Shinde S., Tien D. L., Tople S., Saxena P.: PANOPLY: Low-TCB Linux applications with SGX enclaves. In: NDSS 2017, (2017)

24. Fisch B. A., Vinayagamurthy D., Boneh D., Gorbunov S.: Iron: Functional Encryption using Intel SGX. In: ACM CCS 2017, pp. 765-782 (2017)
25. Sun W., Zhang R., Lou W., Hou Y. T.: REARGUARD: Secure keyword search using trusted hardware. In: IEEE INFORM 2018 (2018)