LogSafe: Secure and Scalable Data Logger for IoT Devices

Hung Nguyen, Radoslav Ivanov, Linh T.X. Phan,
Oleg Sokolsky, James Weimer, and Insup Lee
Dept. of Computer and Information Science, University of Pennsylvania, PA, U.S.A.
Email: {hungng,rivanov,linhphan,sokolsky,weimerj,lee}@cis.upenn.edu

***Abstract*—As devices in the Internet of Things (IoT) increase in number and integrate with everyday lives, large amounts of personal information will be generated. With multiple discovered vulnerabilities in current IoT networks, a malicious attacker might be able to get access to and misuse this personal data. Thus, a logger that stores this information securely would make it possible to perform forensic analysis in case of such attacks that target valuable data. In this paper, we propose LogSafe, a scalable, fault-tolerant logger that leverages the use of Intel Software Guard Extensions (SGX) to store logs from IoT devices efficiently and securely. Using the security guarantees of SGX, LogSafe is designed to run on an untrusted cloud infrastructure and satisfies Confidentiality, Integrity, and Availability (CIA) security properties. Finally, we provide an exhaustive evaluation of LogSafe in order to demonstrate that it is capable of handling logs from a large number of IoT devices and at a very high data transmission rate.**

I. Introduction

With the sharp increase in popularity and use of the Internet of Things (IoT), these devices are bound to accumulate extensive information about users and their activities. The diverse and ad-hoc nature of IoT networks, however, raises concerns both about the privacy and security of the collected data. As shown in several cases, it is possible to uncover private information (e.g., living habits and home addresses) from IoT data through simple appliances such as Nest thermostats [1] and Sharx security cameras [2]. Moreover, IoT devices themselves are vulnerable to various cyber-physical attacks as they are deployed without proper security measures (e.g., Mirai botnet attack [3], SSHowDowN proxy attacks [4]).

As securing all IoT devices is not feasible, it is essential to be able to perform forensic analysis on the IoT data. Such analysis would ensure both that IoT data is managed securely and that attacks on IoT devices are detected and addressed. A necessary condition for such accurate forensics is secure data collection, from a potentially large number of devices, in a tamper-evident and fault-tolerant fashion. In particular, end-to-end security guarantees must be provided, beginning from the communication protocol and ending with secure data storage that provides defenses against unauthorized accesses and attacks. Therefore, this paper addresses the problem of designing a secure and scalable logger for IoT devices.

To support large-scale data logging, cloud service providers such as Amazon, Google, IBM, and Microsoft have emerged as services that allow users of IoT devices to store and manage their data on a cloud infrastructure. The ease of use and integration with other well-known computation and storage services on the same platform is one of the best selling points for these providers. However, there have been several recent security breaches on online services such as Yahoo [5], Ashley Madison [6], and Equifax [7] that potentially affect hundreds of millions of users. Moreover, for proprietary reasons, the precise architecture of these commercial services is not known, which makes it difficult to evaluate their security guarantees.

Without using cloud services, users can choose to deploy logging infrastructure themselves and adequately manage their services. This approach poses the challenge of scalability as administrating a large-scale system needs corresponding resources and expertise. In addition, users can only benefit from running their own infrastructure if the security guarantees are equal or better than cloud services, which are not usually available for the regular consumers (e.g., controlled physical access).

To leverage the advantages of both approaches, namely use the cloud infrastructure but also maintain control of the system's security, we propose LogSafe, a scalable, fault-tolerant logger that provides secure storage of IoT data. Designed to run on the cloud infrastructure, LogSafe is a decentralized logging architecture using Intel Software Guard Extensions (SGX) and standard industrial protocols to guarantee tamper-resilience. SGX is a set of new instructions and memory access changes in Intel architecture design that allows the creation of an enclave, a trusted and isolated execution environment. Enclaves enable applications to maintain confidentiality even when an attacker has physical control of the host machine. Therefore, users can utilize cloud service provider infrastructure and services without worrying about the security and privacy of recorded data. LogSafe also provides the capability to perform online computation on logged data while preserving privacy, e.g., for the purpose of audit or attack detection.

The secure logger presented in this paper is required to satisfy the Confidentiality, Integrity, and Availability (CIA) model, a widely adopted model for assessing the security properties of a given system. Building such a logger in a distributed fashion on the cloud presents several challenges. LogSafe employs Transport Layer Security (TLS) with hashchaining logging scheme [8] and digital signature scheme to guarantee confidentiality and integrity properties. However, even with hashchaining, integrity can be still violated by a replay attack, where an eavesdropper gets a copy of legitimate stored data and at a later time overwrites the current data.

In order to defend against replay attacks, we use the SGX physical monotonic counter to maintain the latest system state as a trusted platform storage. As shown in Section VI, monotonic counter operations are computationally costly and can significantly reduce the system performance if frequently used. To deal with this challenge, we introduce a snapshot algorithm utilizing the SGX counter to defend against replay attacks without compromising the scalability and performance of the system. Specifically, LogSafe uses a fast, secure in-memory counter for node run-time verification and the slower, permanent counter for long-term verification. This combination allows logged data to be verified at any time regardless of system topology changes.

Availability presents its own challenges that arise due to communication overhead (e.g., encryption, gossip protocol) and computation overhead (e.g., cryptographic operations). LogSafe minimizes the performance hit by employing a distributed architecture with decentralized SGX-enabled nodes that can quickly scale up to support a large number of IoT devices. Nodes are organized in a fault-tolerant ring structure (where each node is backed-up by one or two other nodes), and each IoT device is mapped to a specific node based on a consistent hashing algorithm. Finally, note that availability can also be violated by a distributed denial-of-service (DDoS) attack – defending against DDoS attacks is an active area of research, so we leave addressing this problem for future work; at the same time, we note that the fault-tolerant architecture of LogSafe ensures that its operation will not be disrupted even if some of its nodes are attacked.

To evaluate the LogSafe implementation, we first investigate the computational overhead of establishing a secure connection between IoT devices and LogSafe. To assess the scalability of the design, we vary the number of nodes in the system from one to three and observe the change in average processing time per incoming message. Finally, we compare the performance of LogSafe with a non-SGX implementation as well as with previous SGX-based implementations that use a single node [9], [10]. The results indicate that LogSafe is scalable to support a large number of devices for fast logging with very reasonable computation overhead.

To summarize, the contributions of this paper are three-fold: 1) the design and implementation of LogSafe, a cloud-based secure, scalable, and fault-tolerant logger that can accommodate a large number of IoT devices; 2) a scalable snapshot algorithm to defend against attacks without compromising the logger's performance; 3) performance evaluation of the logger implementation to test its scalability properties.

The remainder of this paper is organized as follows. We discuss related work in Section II. In Section III, we present a system overview and provide the problem statement. Section IV then discusses the design architecture of the secure logger and explains the components of the design. We discuss the implementation of LogSafe in Section V. Finally, we present the experimental results of LogSafe in Section VI and provide concluding remarks in Section VII.

II. Related Work

Protecting valuable data from adversaries is an active area of research, and there has been a significant amount of work done on developing tamper-proof loggers for storing information. Secure loggers can generally be classified into those that implement security in software and assume the underlying hardware is attack-free or the ones that implement a secure hardware. Works that use the former design [11]–[14] rely on published commitments to enable tamper-proof service and require a gossip protocol for distribution, which do not scale well in IoT environment with massive scale and spontaneous interaction between devices. On the other hand, works that implement the latter design [15]–[17] use the Trusted Platform Module (TPM) [18] as a trusted computing base to guarantee tamper-proofness. These techniques either assume an adversary who cannot perform sophisticated hardware attacks (e.g., probing memory or launching side-channel attacks) or rely on changing the TPM specifications to provide their guarantees.

Some recent designs [9], [19] propose the use of SGX to defend against a stronger adversary capable of active attacks on the system, leveraging the higher computation capability of SGX in comparison with the previous trusted hardware platforms. Like LogSafe, they maintain a hashchain of logging states as a secure timeline and use SGX to provide a trusted execution environment. Unlike LogSafe, however, these designs are solely for a single computer deployment without fault-tolerant guarantees. SGX-Log [19] encrypts data using SGX sealing feature and can only be decrypted using the same processor, which is unlikely to happen in the cloud environment. Moreover, previous designs rely on SGX monotonic counter (known to be very slow due to interactions with non-volatile memory) for frequent operations, and it is challenging to scale these approaches to the IoT space where interactions with millions of devices might occur. LogSafe does not require SGX monotonic counter for normal operation; it also fully utilizes the cloud infrastructure with distributed architecture to provide scalability and fault tolerance.

SGX has been suggested as a means to build secure systems from an untrusted cloud in a number of works. Haven [20] is the first system to provide a shielded environment to execute legacy Windows applications using SGX. Secure Linux containers such as SCONE [21], Graphene-SGX [22], or Panoply [23] can also be used to run unmodified applications in a trusted enclave. These systems provide greater flexibility for an application running inside the enclave at the cost of higher overhead to encapsulate system functionalities. As the memory available inside the enclave is limited (about 98MB with the current SGX hardware version), this overhead will affect the scalability of the system. LogSafe is designed with the specific logging functionality by executing only core cryptographic operations inside the enclave and proved to be scalable and fault-tolerant to support a high number of IoT devices.

Other works also propose SGX for secure cloud analytics such as VC3 [24] with secure MapReduce computations or

IRON [25] with secure functional encryption. These algorithms can be used in conjunction with LogSafe to provide multi-party analytics over logged data.

III. Overview of the IoT Logging Architecture

This section first introduces the background of Intel SGX and gives an overview of the proposed system with the security properties such a system should provide. Then, it outlines the possible attack surfaces that could affect those properties as well as our approach to ensure that the system does provide them.

A. Intel SGX Background

SGX was designed by Intel in order to address the problem of executing software applications in a remote computer owned by an untrusted party, while at the same time providing integrity and privacy guarantees [26]. At a high level, SGX is a set of new instructions and memory access changes in the Intel architecture; it works by instantiating an enclave at the remote computer for the purposes of computation and information exchange. The enclave is essentially a secure, separated and encrypted region for code and data are decrypted only inside the processor.

The enclave enables applications to maintain confidentiality even when an attacker has physical control of the platform. Special CPU instructions, such as `EENTER` (to execute the code inside the enclave) and `EEXIT` (to quit execution), must be used by the enclave's host process to interact with the enclave, and it happens in protected mode. Exceptions are raised when a non-enclave access to a memory is attempted by a software, and also when a code fetch is attempted from inside an enclave to an address range outside that enclave. Therefore, SGX ensures that the secure regions of code and data are able to maintain confidentiality even when an attacker has physical control of the platform and can conduct direct attacks on memory.

Each SGX-enabled processor is fused with a specific key during the manufacturing process that can be used to encrypt and integrity-protect sensitive data. To store the secret on untrusted memory, an enclave program can call `EGETKEY` instruction to derive an encryption key from the persistent hardware-based key. This encryption key can only be retrieved by instances of the same enclave program on the same platform. This sealing feature ensures that sensitive data are isolated between enclaves or even between different versions of the same enclave program (e.g., an older vulnerable version cannot access sealed data from a newer version).

SGX also provides a remote attestation mechanism to allow another party to verify that the correct program is securely running within an enclave on the remote platform. An enclave can use the `EREPORT` instruction to generate an unforgeable report containing information to verify the trustworthiness of the enclave and the platform. The report is signed using a private key for Intel Enhanced Privacy ID (EPID), an anonymous group signature scheme. The party can validate the report and contact Intel Attestation Server (IAS) to verify the signature of the report.

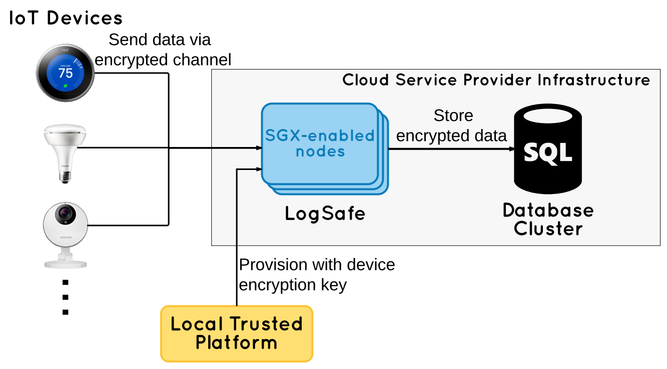

IoT Devices
75
Send data via encrypted channel
Cloud Service Provider Infrastructure
SGX-enabled nodes
Store encrypted data
SQL
LogSafe
Database Cluster
Provision with device encryption key
Local Trusted Platform

Fig. 1: The LogSafe system overview.

The SGX security is formally proved in [25]. However, SGX also has limitations, especially vulnerability against side-channel attacks [26]–[29]. Therefore, it is the responsibility of the enclave program implementation to defend against side-channel attacks. Fisch et al. [25] also present different techniques to defend against such attacks.

B. System Overview

Figure 1 shows the logger's intended environment. LogSafe is to be deployed in a vast IoT network that may potentially contain a very large number of devices (ranging in the billions according to some current estimates [30]). Since IoT devices are generally resource constrained devices (e.g., fitness tracking devices, medical devices) that do not have the capacity to perform computation-heavy operations or to store great amounts of data, the logger's task would be to securely store all data generated by these devices as well as to provide a platform to compute certain functions on the logged data (e.g., sensor attack detection) without revealing any information to unauthorized entities.

Due to the unprecedented size of IoT networks, developing and managing the infrastructure for such a logger system would be a challenging task. That is why, in this work we propose to leverage the existing cloud infrastructure and augment it with the security features of SGX. In particular, each machine on the cloud would be capable of running an SGX enclave;[1] in turn, the security guarantees provided by SGX, namely remote attestation and secure computing, would make it possible to securely execute code on the cloud without exposing information to the cloud provider or the rest of the world.

In order to exploit the full capacity of the cloud, LogSafe is distributed as well (refer to Section IV for a description of the specific design of the distributed logging system). This allows us both to handle a larger number of incoming messages and to provide an extra layer of fault/attack tolerance as opposed to a centralized system with a single point of failure.

[1]This is a reasonable assumption considering that all new generations of Intel desktop/laptop chips and some server chips are capable of running SGX.

In case a machine crashes or experiences denial of service, LogSafe would transfer its state and responsibilities to a back-up machine, thus ensuring the seamless execution of the entire system.

In this framework, a single trusted authority platform (running locally) can provision an arbitrary number of SGX-enabled nodes running on the cloud infrastructure. Once an IoT device sends (encrypted) data to the logger node, LogSafe verifies the authenticity of the data, encrypts it with the provisioned encryption key and sends it to the cloud database service (e.g., Amazon RDS) for permanent storage. This ensures that data is only decrypted within an SGX enclave and cannot be accessed from the outside. In a similar fashion, LogSafe provides a platform for secure computation as well – if a user would like to perform computation on the logged data, an SGX enclave would retrieve the data from the database, decrypt it and securely perform the required computation, only returning the answer to the user (e.g., saying that no attacks were detected).

C. *Security Discussion*

Due to its application domain, namely a large IoT network that is constantly subjected to outside threats, LogSafe is exposed to a number of attack surfaces. In this work, we focus on building a system that satisfies the CIA model, a standard model in the cyber security domain. We utilize different techniques in order to (attempt to) achieve each of the three CIA properties, as discussed below.

Confidentiality means that IoT data cannot be seen by unauthorized entities (e.g., through an eavesdropping attack). LogSafe attains this property by employing the TLS protocol; TLS provides end-to-end security between the devices and LogSafe by ensuring that messages are encrypted, hashed, and signed. Thus, by making the standard assumptions about the hardness of prime number factorization, we can make sure that IoT data is only readable inside an SGX enclave.

Integrity is achieved when the data that is logged (and retrieved later) is the same as the data that was originally sent by an IoT device. Integrity can be violated in systems vulnerable to injection attacks where the attacker is able to modify the data stored in the database. LogSafe defends against injection attacks by using a hashchain [8] algorithm in order to ensure that the logged data is consistent and in the right order. Even in this case, integrity can also be violated by a variant of a replay attack in which the attacker replaces the latest logged data with a previous authentic snapshot – in this case the data is still consistent but is incomplete. In order to address this issue, we use the monotonic (physical) counter provided by SGX, which serves as a checkpointing mechanism; by verifying that the counter stored on the database is equal to the one in SGX, we can ensure that the data that was last received is indeed stored on the database.

In addition to attacks on the database itself, attacks on IoT devices can also compromise the integrity of the logged data. In particular, since most IoT devices are not developed with security features in mind, they might be easily corrupted and, consequently, transmit wrong data to LogSafe. To detect such scenarios, LogSafe provides a platform for secure computation where existing attack detection techniques [31] can be executed inside an SGX enclave without revealing any data to the outside world, except for the output of the computation.

Finally, availability holds when the system is able to handle all messages and requests that it receives. Availability is difficult to guarantee in the worst case because there are no known fail-safe techniques for defending against DDoS attacks. At the same time, by developing a distributed system on the cloud (possibly on multiple clusters), we can alleviate the effect of DDoS attacks by requiring attackers to compromise many more machines in order to disrupt the functionality of LogSafe.

With the above considerations in mind, we can now concisely state the problem addressed by this paper and LogSafe.

Problem: *This paper addresses the problem of how to design and implement a distributed cloud-based logger for IoT devices using SGX. The logger must satisfy the CIA properties in the presence of eavesdropping, injection, and replay attacks.*

IV. Design of LogSafe

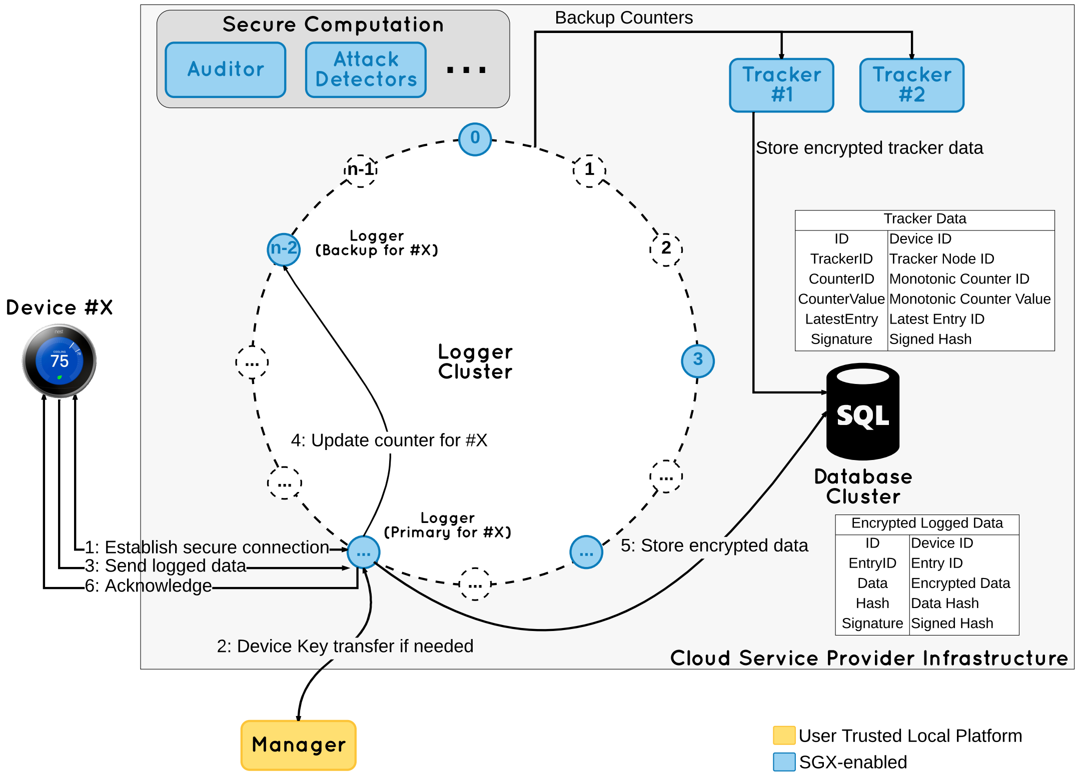
In this section, we describe the architecture and the various components of the logger design (for easy reference, the architecture diagram is presented in Figure 2). Note that our design makes the following (standard) assumptions that ensure the logger security properties are satisfied:

- SGX is implemented correctly and not compromised,
- cryptographic primitives such as RSA and Advanced Encryption Standard (AES) encryption are safe,
- an attacker cannot forge digital signatures, and
- the hash function is collision-resistant.

A. *High Level Data Flow*

This subsection describes at a high level the data flow within LogSafe during its operation; the specifics of each phase are described in their corresponding subsections of this section. The IoT device initiates the protocol by establishing the secure connection with the *Logger*. After authenticating each other (i.e., validate the certificates), the *Logger* checks if it already has the required information to process requests from the IoT device. If not, it requests provisioning from the *Manager* to acquire needed device's meta-data and seal this information for further usage (Step 2 in Figure 2); the *Manager* is described in more detail in Section IV-D.

Upon receiving the data message, the *Logger* enclave first calculates $entry\_id$, a monotonically increasing sequence number that is incremented after each message and is stored in the volatile memory of each *Logger*. Then the *Logger* encrypts the received data using the device's encryption key provided by the *Manager* and generates the next hashchain block following the tamper-evident logging scheme. As discussed in Section IV-E below, hashchaining ensures that injection attacks can be detected by storing a consistent hashed version of all logged data; this hash also contains $entry\_id$ such that the order of message arrival can also be verified in the hash chain.



Tracker Data	
ID	Device ID
TrackerID	Tracker Node ID
CounterID	Monotonic Counter ID
CounterValue	Monotonic Counter Value
LatestEntry	Latest Entry ID
Signature	Signed Hash

Encrypted Logged Data	
ID	Device ID
EntryID	Entry ID
Data	Encrypted Data
Hash	Data Hash
Signature	Signed Hash

Fig. 2: LogSafe architecture and data flow.

On the cloud platform, nodes are expected to fail or crash – in such a scenario, the crashed node cannot provide *entry_id* for log verification. Therefore, before making any changes in the persistent storage, the *Logger* will send the latest *entry_id* to its successor node to back up the latest state of this device (Step 4 in Figure 2). To ensure consistency, LogSafe only processes requests from a device sequentially.

Even with backup nodes, all the device states are still kept in the volatile memory and would not be available after the cluster is shut down. Therefore, nodes occasionally send device states to Trackers for a snapshot backup. The snapshot algorithm (described in Section IV-F) is essentially another hash chain log of the device states, but more importantly, it uses SGX counters (non-volatile memory) for version tracking. To verify if an IoT device's log is the latest version (replay attack detection), the auditor can query the corresponding node serving the device if it is still active. Otherwise, it can trace back the snapshot data and query the tracker's SGX counter to validate the freshness of the snapshot.

In the following subsections, we first provide the features that support LogSafe architecture. After that, we refine the high level data flow into a practical system:

- LogSafe consists of a decentralized distributed cluster (Section IV-B) to guarantee high-availability and fault tolerance.
- IoT devices need to establish a secure connection following a handshake protocol (Section IV-C) with LogSafe before being able to send log. Both IoT device and LogSafe must be authenticated during the protocol, which might require provisioning (Section IV-D) from the local trusted authority.
- LogSafe requires both fast, in-memory counter (Section IV-E) for node run-time verification and slow, permanent counter (Section IV-F) for long-term verification.
- LogSafe also provides a secure computing platform (Section IV-G) for log auditing and attack detection.

B. Distributed Logger Cluster

The heart of LogSafe is a decentralized cluster of *Logger* nodes, i.e., there is no centralized control or hierarchical organization between the nodes. The *Logger* cluster uses Chord [32] as the distributed look up protocol, where given each IoT device is mapped to a *Logger* node using the device's id as key. Chord uses a consistent hashing algorithm to efficiently assign IoT devices to the corresponding nodes. The consistent hashing also reduce device assignment movement when nodes are added to and removed from the system.

Availability is the main reason LogSafe is designed in the distributed architecture. The cluster can be seen as an n-node ring (to support consistent hashing), where n is the maximum number of nodes that join the system at any given time. Depending on the workload, the user can choose to add or remove nodes to balance the needs without much effect on the system performance. In addition, LogSafe also provides fault tolerance as each *Logger*'s state (containing information about all IoT devices connected to it) is always replicated in (configurable

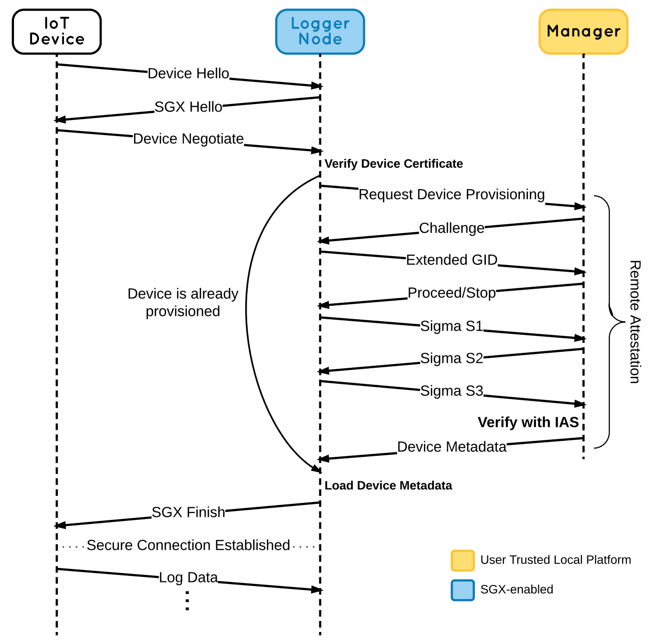


Fig. 3: Handshake protocol.

number) backup nodes. If the primary *Logger* fails, the IoT device can switch to a backup node for continuous operation. The backup node then continues to replicate the state to ensure the replication factor is always met.

Since SGX does not support I/O operations (i.e., every call to the I/O needs to leave the encrypted enclave, and re-enter with the results), LogSafe is managed in a decentralized fashion to avoid a single point of failure with an SGX node handling a high number of I/O requests. There is also a possibility that Logger node's operating system is compromised and affects the network stack. This could at most result in DDoS (i.e., the node looks unresponsive to the IoT device or other nodes). One way to mitigate the problem is allocating nodes in different zones of the data center, or even different data centers to minimize the number of nodes that can be attacked at the same time. As long as LogSafe still has correct functional nodes, it can detect anomalies and re-balance the system.

C. Handshake Protocol between IoT Device and LogSafe

Once an IoT device is mapped to a *Logger* node, the two devices must first establish a secure connection (by following the TLS handshake procedure). Once connection is established, the IoT device starts transmitting data to the logger. A typical message exchange between an IoT device and the logger is presented in Figure 3.

To establish a connection, the IoT device first initiates the handshake protocol by sending a `Device Hello` message with cryptographic information such as the TLS version, along with the cipher suites supported by the client. In response, the logger replies with an `SGX Hello` message that contains the chosen cipher suite along with its digital certificate. Upon receiving the device's certificate, the logger first verifies the certificate authenticity, then checks whether the untrusted storage (i.e., the database) already has sealed device meta-data (i.e., encryption key). If the meta-data does not exist, the logger will start provisioning protocol with the manager as described in the following subsection. Otherwise, it will unseal the meta-data into memory and query the latest log entry from the database cluster to restore the hashchain. Finally, the logger finishes the handshake protocol by sending an *SGX Finish* message.

Note that an AES symmetric session key is also agreed upon during the handshake; it is used by the IoT device for data encryption in all communication henceforth.

D. The Manager and Provisioning Protocol

LogSafe requires a local trusted platform to function as the *Manager*, responsible for provisioning both the IoT devices and the logger nodes running on the cloud. The idea is that the LogSafe user can run a single *Manager* node (even on a non-SGX machine) locally in order to be able to manage LogSafe on the cloud infrastructure.

Each IoT device can be provisioned during the manufacturing process with a unique identifier id, a private/public key pair (pk_d, sk_d), the corresponding trusted root certificate, and the Tracker's addresses. This key pair is only used for establishing a secure TLS connection with the logger node. In addition, an encryption key sk_{enc} is also generated for the device, but this key is stored only in the *Manager* for further provisioning with the logger node.

LogSafe can have an arbitrary number of nodes running on the cloud. Each *Logger* instance is essentially an untrusted application, which loads the signed binary of the logger enclave program. As these *Logger* nodes are managed by the cloud service provider, they need to be correctly verified before the *Manager* sends any sensitive information to the nodes, e.g., the key pairs (pk_l, sk_l) to establish the TLS connection with IoT devices, the key pairs (pk_{sign}, sk_{sign}) to sign encrypted log entry, and device encryption key sk_{enc}. The verification process can be done via SGX remote attestation as shown in Figure 3 and detail below.

To initiate remote attestation, *Logger* sends a requests to receive a random challenge from *Manager* (to prevent session reuse). Next, the *Logger* sends the Extended Group ID (GID) of EPID to the *Manager* to be verified, followed by a modified Sigma protocol (i.e., involves three-step structure: commitment, challenge, and response – showing as `Sigma S1`, `S2`, and `S3` in Figure 3), during which *Logger* and *Manager* also perform a Diffie-Hellman Key Exchange for secure communication. After receiving `Sigma S3`, which contains the report from the *Logger*, the *Manager* can contact IAS server to validate the authenticity of the report. If the *Logger* is verified, it will received provisioning information from the *Manager* via the established secure connection.

E. Tamper-evident Logging Scheme

Once the initial handshake is complete and LogSafe has received the data message, it initiates the logging procedure.

Algorithm 1 Logging Algorithm

```
Logger State:
    $entry\_id_{k-1}$ ← previous log entry ID //($id_0 = 0$)
    $h_{k-1}$ ← previous hash value //($h_0 = 0$)
Input:
    $id$ ← device ID
    $c_k$ ← current message content
Parameters:
    $sk_{enc}$ ← device encryption key
    $sk_{sign}$ ← logger private key

1: procedure AddEntry($id$, content $c_k$)
2:     $entry\_id_k \leftarrow entry\_id_{k-1} + 1$
3:     $enc\_data_k \leftarrow$ Encrypt($entry\_id_k || c_k, sk_{enc}$)
4:     $h_k \leftarrow hash(h_{k-1} || entry\_id_k || c_k)$
5:     $sh_k \leftarrow sign(h_k, sk_{sign})$
6:     BackupCounter($id, entry\_id_k$)
7:     Write($id, entry\_id_k, enc\_data_k, h_k, sh_k$)
8:     Acknowledge($id$)
```

Algorithm 2 Snapshot Algorithm

```
Tracker State:
    $s_{k-1}$ ← previous sequence number //($s_0 = 0$)
    $h_{k-1}$ ← previous hash value //($h_0 = 0$)
Input:
    $id$ ← device ID
    $entry\_id_k$ ← latest entry counter from the device
Parameters:
    $tracker\_id$ ← tracker ID
    $sk_{sign}$ ← tracker private key
    $counter\_id$ ← monotonic counter ID

1: procedure AddSnapshot($id$, $entry\_id_k$)
2:     $s_k \leftarrow s_{k-1} + 1$
3:     $h_k \leftarrow hash(h_{k-1} || id || entry\_id_k || s_k)$
4:     $sh_k \leftarrow sign(h_k, sk_{sign})$
5:     Write($tracker\_id, counter\_id,$
               $s_k, id, entry\_id_k, h_k, sh_k$)
6:     Acknowledge($id$)
```

The complete logging scheme is presented in Algorithm 1. Upon receiving a new data message, the logger enclave first calculates the entry ID $entry\_id_k$, a monotonically increasing sequence number and is incremented after each message. Given $entry\_id_k$, the logger now encrypts the received data, concatenated with the sequence number (Line 3 in Algorithm 1); the data is encrypted using the corresponding IoT device's encryption key, as provided by the *Manager*.

In addition, for tamper detection reasons, we compute the hash h_k of all the information related to the current message (sequence number and the message), together with the hash of the previous message h_{k-1} (Line 4 in Algorithm 1). The hash h_k is then signed by the logger enclave using the enclave's private key sk_{sign} (Line 5 in Algorithm 1). As discussed in [8], this hash chain ensures that the proper message sequence is preserved, such that any attempt to change message content will create a different hash chain branching off at the affected log entry, thereby enabling us to detect the data tampering.

In summary, for each device the logger stores two classes of information – the encrypted data (encrypted using the corresponding device's encryption key) and a hash of the data, together with the hash of previous message. This hash chain ensures the tamper-evident property of the logger by detecting any data modifications by external entities (e.g., the attacker modifying the contents stored on disk).

F. Counter Snapshot Protocol

The mechanisms described so far ensure that stored log can be appropriately verified when *Logger* node is running. We also need to make sure that the security properties still hold even when user shuts down the cluster. One way to achieve this is having each node seal the system state upon shutdown. This approach might not be suitable for the cloud environment where physical machines are usually assigned to different users depends on priorities and workload.

LogSafe makes sure long term stored log can still be verified via the use of the *Tracker* node for snapshots. This node can be replicated (by a factor of two in Figure 2) to ensure fault-tolerance. *Tracker* functions very similarly to *Logger* node, with two main differences: (1) instead of creating logs for IoT devices, it creates snapshots for device latest counter values; (2) *Tracker* uses SGX physical monotonic counter instead of memory variable.

SGX monotonic counters are available via Platform Service Enclave, with up to 256 counters available [33], identified by a counter ID and a nonce. Each operation with the counter is performed on non-volatile memory such that an enclave with the same signing key can access the counter to read/increment even after the machine is shut down/rebooted.

With the SGX monotonic counter, the snapshot algorithm is presented in Algorithm 2. In addition to the use of physical monotonic counter and different generated content, the snapshot algorithm just creates the hashchain and signature without encrypting the data. The reason is these information are already available in plaintext on *Logger* storage. We only need to guarantee the authenticity and freshness of the data stored in snapshots. Since the operations with monotonic counter are very slow, we aim to minimize snapshot creation in specific cases: logger cluster shutdown, IoT device inactive for a fixed period of time (e.g., *Logger* node can temporary free memory for other devices).

In summary, given both in-memory *Logger* counter and permanent *Tracker* counter, LogSafe allows logged data to be verified at any time regardless of system topology changes.

G. Secure Computation

Although the logging scheme presented in the previous subsections provides tamper evidence if the logged data have been changed, it does not address the case where the IoT device itself might be under attack, possibly transmitting wrong information for storage. In order to handle such a scenario, LogSafe also supports delegating secure computation (e.g., attack detection) on the logged data. The idea is very similar to IRON's secure functional encryption [25], such that functional enclaves can be deployed to the cloud to securely decrypt the sensitive data, perform the prescripted computations, and only return the result to the user. Note that system designers need to approve any such computation before it is allowed to run on the enclave (in other words, arbitrary computation is not allowed because a malicious node might ask for all the data, for example).

There are various techniques can be used to detect possible IoT attacks, such as model-based approaches [34], sensor and information fusion [31], [35], as well as data-driven machine learning approaches [36]. These algorithms, along with the hashchain verification algorithm [9] can be implemented as query primitives and provisioned to nodes on the cloud by the Manager. It can perform remote attestation to ensure the node is correctly started before giving decryption key to the node so that the cloud services can regularly execute audits for attack detection.

V. Implementation

We implemented the LogSafe prototype in C++ with four applications: IoT *Device*, *Logger*, *Tracker*, and *Manager*. These applications represent the main modules that constitute the LogSafe architecture. The *Device* was developed on both Windows 10 Professional and Ubuntu 16.04 without any dependency to SGX (since it is potentially untrusted). All other modules were developed on Windows 10 Professional using the Intel SGX SDK 1.8 for Windows[2] (currently, Intel SGX SDK for Linux does not support all the required features such as monotonic counter operations).

The applications share the same untrusted library that was developed using Boost Asio 1.65.1[3] to provide asynchronous event-driven support across LogSafe. In addition, the *Logger* and *Tracker* applications also load the corresponding enclave dynamic libraries, `logger.dll` and `tracker.dll` respectively. These libraries feature trusted functions (ECALLs) to be executed inside the enclave, while the applications provide untrusted functions (OCALLs) to be called from the enclave.

Conceptually, the *Device* exposes a synchronized `AddLog` function, during which it establishes a secure connection with the *Logger* (if it does not already exist) and sends the message over this connection. The *Logger* and the *Tracker* share the same code base but have different event handlers to feature different purposes. After loading the enclave libraries, they start to listen on the pre-defined internal port for incoming LogSafe messages. The *Logger* also listens on additional service port for incoming IoT device's requests.

LogSafe uses cryptographic primitives from SGX SDK's trusted cryptographic library `sgx_tcrypto.lib` and OpenSSL library 1.0.2 (with trusted version[4] for the enclave libraries and untrusted version[5] for the applications). More specifically, LogSafe encrypts data with Rijndael AES-GCM encryption on 128-bit key size and 96-bit initialization vector. Logged data are hashed using SHA256 and signed using 256-bit elliptic curve digital signature algorithm (ECDSA). SGX SDK also provides 256-bit elliptic curve Diffie-Hellman key exchange algorithm (ECDHE) to establish the secure connection during remote attestation process. These cryptographic primitives are provided with resilience to side-channel attacks.

LogSafe makes use of two optimizations to reduce computation costs on IoT devices:

1) Front-end router: With Chord's lookup protocol, an IoT device only needs to know the subnet of LogSafe cluster and finds an active node to lookup for its primary *Logger* node. To further reduce the time needed for the device to find the correct logger node to start the logging procedure, LogSafe has a non-SGX *Router* as the front-end of the system. Each logger node periodically sends a heartbeat message to the *Router* to keep the alive status in the node list. Upon receiving a request from an IoT device, it will be able to compute the address of the corresponding logger node and return the information to the IoT device. It is important to mention that the *Router* is not trusted, which means it can return the wrong information or not reply anything. However, as the correctness of the matching will be verified during the handshake protocol, the IoT device will eventually find the correct node following Chord's lookup protocol.

2) TLS session resumption: In IoT applications where devices often move around and use unreliable Internet connectivity, these devices may need to re-establish secure connection frequently. This communication overhead can be very costly for IoT devices as shown in Section VI. LogSafe can improve the TLS performance by keeping the established session state (represented by OpenSSL's `SSL` and I/O stream abstraction `BIO` objects) so that an existing device can reuse the previous session when reconnecting to LogSafe. However, a session cannot be used forever to ensure forward secrecy. IoT device can only reuse a session up to a configurable time before the device needs to negotiate a new session. The session state is kept in the enclave and time is measured using the trusted clock from SGX platform services to ensure security guarantees.

VI. Evaluation

In this section, we present the experimental evaluation of the LogSafe implementation. We first discuss the overhead for setup time before presenting the scalability of LogSafe in normal operation and in snapshot operation. Finally, we

[2]https://software.intel.com/en-us/sgx-sdk/download

[3]http://www.boost.org/doc/libs/1_65_1/doc/html/boost_asio.html

[4]https://software.intel.com/sites/default/files/managed/3b/05/Intel-SgxSSL-Library-User-Guide.pdf

[5]https://www.openssl.org/source/

TABLE I: Execution time breakdown of handshake protocol. Remote attestation is needed only if the *Logger* does not have the device's meta-data and requires provisioning.

Intel Edison	Dell 5480	Task	Logger
$717\mu s$	$87\mu s$	Device Hello $\rightarrow$	
		$\leftarrow$ SGX Hello	$5{,}886\mu s$
$370{,}055\mu s$	$8{,}617\mu s$	Device Negotiate $\rightarrow$	
		(*) Remote Attestation	$1.038s$
		$\leftarrow$ SGX Finish	$5{,}935\ \mu s$
$1.420s$	$1.059s$	Total time (with remote attestation)	
$382.5ms$	$20.5ms$	Total time (without remote attestation)	

compare LogSafe with previous proposed loggers using both SGX and non-SGX implementations.

A. Setup Time

Before any IoT device begins to send encrypted data to LogSafe, it needs to establish a trusted connection with the *Logger* enclave as described in Section IV. In this experiment, we measure the setup time needed until both the IoT device and LogSafe successfully establish a TLS channel in two cases: (1) an IoT device connects to LogSafe for the first time (i.e., *Logger* needs to request provisioning with the *Manager*); (2) an IoT device reconnects to the *Logger* (i.e., this node already has the device's meta-data sealed).

In the experimental setup, the *Logger* and the *Manager* are installed on two Dell Latitude 5480 laptops, each with a 2.5GHz Intel Core i5-7200U CPU and 8GB RAM, running Windows 10 Pro. The client application is executed on an Intel Edison board, a computing module with a 500 MHz Intel Atom and 1GB RAM, running Yocto Linux. All the parties are configured with TLS 1.2 and `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384` cipher suite. To quantify the impact of the specific IoT device platform on setup time, we also run the client application on another Dell Latitude 5480 laptop and compare the two times. All the machines are connected via a 1Gbps switch while the Intel Edison board is connected via a Wireless Access Point 802.11n 150Mbps on the same switch.

As can be seen from Table I, the setup time is significantly higher if the *Logger* needs to be provisioned. As described earlier, the remote attestation procedure not only involves a communication protocol between the cloud node and the *Manager*, but it also requires communication with Intel IAS server. The current version of IAS requires establishing a TLS session with a valid client certificate before any API calls can be made. In our experiment, it takes almost 600ms to get the results from Intel IAS server. However, it is important to emphasize that *Logger* provisioning is a one-time cost and does not affect the long-term performance of our system.

RSA-involved operations (i.e., *SGX Hello*, *Device Negotiate*, and *SGX Finish*) need almost equivalent time on both sides running on Dell laptop. *Device Negotiate* requires a slightly higher time because the IoT device needs to not only verify the certificate sent by LogSafe, but also generates its corresponding certificate reply. However, without provisioning, the overhead is clearly dominated by *Device Negotiate* on the lower-computation-power Intel Edison. The results further confirm the benefits of the TLS session resumption feature as the overhead only takes place at the beginning of each session (e.g., once every hour, as in a standard TLS configuration), so it is not expected to present a computational burden.

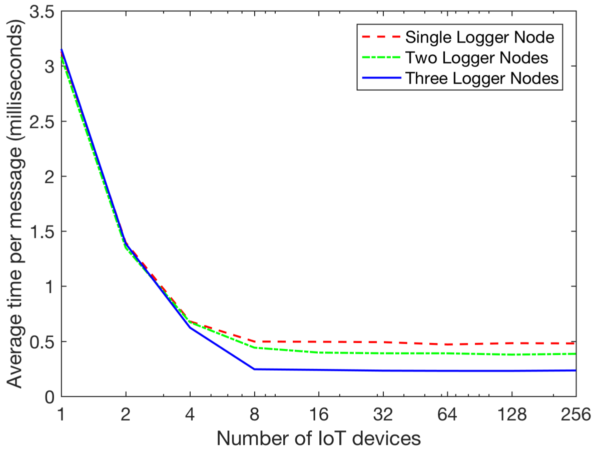


Fig. 4: LogSafe's average message processing time with different configurations under variant workloads.

B. Logging Performance

Adding security guarantees using SGX increases message transfer latency. To quantify the impact and evaluate the benefits of the distributed logger, we set up an experiment to measure the average message processing time. This experiment stress tests LogSafe under different cluster configuration: (1) a single Logger node (with another node functioning as backup only); (2) two fully functional Logger nodes (i.e., two nodes processing requests from IoT devices and acting as backup for each other); (3) three fully functional Logger nodes. We use Dell Latitude 5480 machines to run the cluster, while another machine is used to synthetically generate requests (100 bytes messages) from IoT devices, ranging from 1 to 256 devices.

Figure 4 shows the results. With only one IoT device, there is no difference between three configurations with an average of 3ms per request. This can be explained by the communication flow in Section IV: LogSafe processes requests from a device sequentially because the device must receive an acknowledgment message before sending the next requests. The multi-threading implementation starts to benefit with two IoT devices joining the system as the requests can be processed in parallel, resulting in half processing time.

As the number of IoT devices increases beyond the number of cores on the single- and double-node systems, the average processing time for the cluster with three nodes is roughly an order of magnitude lower (as low as 0.2ms per request). We also notice that at best performance, the gain from one-node to other configurations is not double or triple based

TABLE II: Execution time breakdown of cryptographic functions. Snapshot algorithm only involves hashing, signing, and counter increment.

Operation	Message Size (bytes)									
	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}
Encryption	0.30 μs	0.30 μs	0.40 μs	0.40 μs	0.50 μs	0.70 μs	1.20 μs	2.10 μs	3.80 μs	7.50 μs
Hash	0.30 μs	0.50 μs	0.80 μs	1.20 μs	2.30 μs	4.20 μs	7.90 μs	15.70 μs	31.00 μs	60.70 μs
Sign	388.92 μs									
Counter Increment	152 ms									

on the number of nodes. It is because these nodes not only process incoming requests from IoT devices, they also need to function as backups for other nodes. Nevertheless, the results illustrate the benefit of the scalable design of LogSafe – for any fixed number of IoT devices, the average processing time per message is bound to greatly decrease as the number of nodes in the LogSafe cluster increases. To provide a more practical interpretation of the above results, note that if IoT devices send data every 1 second, the three-node LogSafe can support 5000 devices simultaneously.

C. Cryptographic Operation Latency

In this subsection, we measure the effect of the proposed snapshot algorithm, in which the SGX monotonic counter is periodically incremented; each counter increment is expected to be slow due to non-volatile memory interactions. To quantify this and identify the bottleneck of cryptographic operations, we measure the latency of each cryptographic operation, including encryption, hashing, signing, and counter increment; for more exhaustive evaluation, the message size is varied from 2^5 bytes (AES block size) to 2^{14} bytes (maximum TLS plain text length specified by TLS specification [37]). Note that the hashchaining algorithm only signs concatenated hashes and counter values, which have fixed length, so that the time required for sign operation does not change with message length.

Table II provides the average time it takes to perform the logging sub-procedures. In particular, the time needed for encryption and hashing increases linearly with the message length. Incrementing the SGX-based counter takes almost 152ms and signing takes 389μs on average, which are much higher than other operations, as expected. This has a significant effect on snapshot time, resulting in approximately 160ms per snapshot processing time. However, the typical snapshot data are only dozen bytes containing the device ID and latest counter value. In addition, it is expected that *Tracker* usually takes snapshot only when a node is shutting down. Thus, the snapshot algorithm is able to minimize the overhead introduced by incrementing the monotonic counter.

D. Performance Comparison

Finally, it is important to note the overhead of LogSafe and discuss the trade-off of using SGX by comparing the proposed logger with one that does not use SGX, and with other proposed secure loggers using SGX. In particular, this

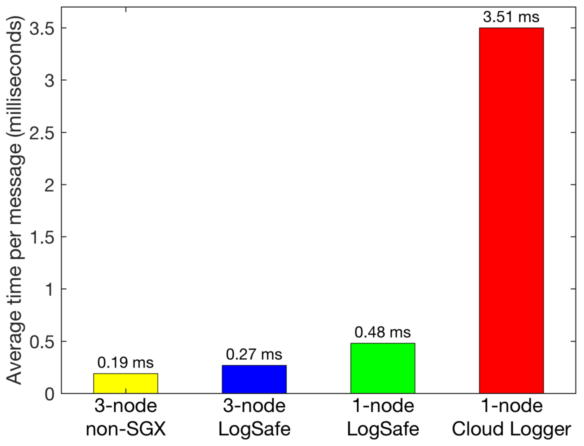


Fig. 5: Average message processing time comparison between LogSafe and other implementations.

experiment aims to measure the average processing time for 100-byte messages by LogSafe in comparison with a logger implementing the same algorithm without using SGX, and with an implementation of a single-node SGX-based logger, referred to as Cloud Logger in this paper, proposed in our prior work [9]. For LogSafe, we use two cluster configurations: single-node and three-node. The non-SGX logger features the same algorithm but was implemented using Intel Integrated Performance Primitives Cryptography library[6] to fully utilize AES-NI instructions. Lastly, the CloudLogger is configured with a buffer size of 50.

The results are illustrated in Figure 5. As expected, the non-SGX logger yields the best performance using the same cluster configuration with LogSafe. At the same time, the overhead of SGX (namely, enclave boundary data transfer and different cryptographic implementations) does not appear to be prohibitive as the average processing time per message is only about 42% slower – furthermore, the difference in average processing times is bound to decrease in systems with more nodes and lower processing times. Thus, LogSafe provides much better privacy and security guarantees, especially over cloud adversary, while paying a reasonable price in terms of performance.

In addition, it can be seen that LogSafe greatly outperforms

[6]https://software.intel.com/en-us/ipp-crypto-reference

Cloud Logger – LogSafe is more than 10 times faster than the one-node Cloud Logger which performs a physical counter increment after every few messages from an IoT device. Note that Cloud Logger is also very similar to SGX-Log [10], another single-node SGX-based logger. However, SGX-Log is executed in Intel SGX SDK for Linux, which does not support the physical counter increment since all current Linux versions are not able to interact with physical monotonic counters and only emulate the counter in software. Hence, SGX-Log effectively provides weaker security guarantees, which is why a fair comparison between LogSafe and SGX-Log is impossible.

VII. Conclusions

In this paper, we described LogSafe, a cloud-based logger for the IoT environment that stores secure logs from devices and allows for forensic analysis in case of an adversarial event. We used SGX as a trusted hardware to enable the design of the logger, which guarantees confidentiality and integrity of stored data and provides tamper-detection of the data. LogSafe is highly scalable and fault-tolerant with decentralized distributed logger nodes that can be provisioned on the fly based on system workload. We leveraged the greater computational power of SGX to improve on works that attempt secure logging and to provide stronger security guarantees. LogSafe is able to defend against three classes of attacks, namely replay, injection and eavesdropping attacks. Finally, based on the simulation experiments, the proposed logger is scalable to support a large number of IoT devices as well as a large transmission of data.

Acknowledgment

This work was supported in part by NSF CNS-1505799 and the Intel-NSF Partnership for Cyber-Physical Systems Security and Privacy and by ONR N000141712012

References

[1] (2016) Nest thermostat leaked user data over wifi. [Online]. Available: https://consumerist.com/2016/01/20/nest-thermostats-have-been-leaking-users-home-locations-over-wifi/

[2] (2016) Princeton researchers find security flaws in iot devices. [Online]. Available: https://www.engadget.com/2016/01/21/princeton-researchers-iot-security-flaws/

[3] (2016) Hacked cameras, dvrs powered todays massive internet outage. [Online]. Available: https://krebsonsecurity.com/2016/10/hacked-cameras-dvrs-powered-todays-massive-internet-outage/

[4] E. Caltum and A. T. R. Ory Segal. (2016) Exploitation of iot devices for launching mass-scale attack campaigns. [Online]. Available: https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/sshowdown-exploitation-of-iot-devices-for-launching-mass-scale-attack-campaigns.pdf

[5] (2016) Yahoo says at least 500 million accounts breached in attack. [Online]. Available: http://www.bloomberg.com/news/articles/2016-09-22/yahoo-says-at-least-500-million-accounts-breached-in-hack-attack

[6] (2015) Online cheating site ashley madison hacked. [Online]. Available: http://krebsonsecurity.com/2015/07/online-cheating-site-ashleymadison-hacked/

[7] (2017) Equifax announces cybersecurity firm has concluded forensic investigation of cybersecurity incident. [Online]. Available: https://www.equifaxsecurity2017.com/2017/10/02/equifax-announces-cybersecurity-firm-concluded-forensic-investigation-cybersecurity-incident/

[8] P. Maniatis and M. Baker, "Secure history preservation through timeline entanglement," *arXiv preprint cs/0202005*, 2002.

[9] H. Nguyen, B. Acharya, R. Ivanov, A. Haeberlen, L. T. Phan, O. Sokolsky, J. Walker, J. Weimer, W. Hanson, and I. Lee, "Cloud-based secure logger for medical devices," in *Connected Health: Applications, Systems and Engineering Technologies (CHASE), 2016 IEEE First International Conference on*. IEEE, 2016, pp. 89–94.

[10] S. Chandra, V. Karande, Z. Lin, L. Khan, M. Kantarcioglu, and B. Thuraisingham, "Securing data analytics on sgx with randomization," in *European Symposium on Research in Computer Security*. Springer, 2017, pp. 352–369.

[11] B. Schneier and J. Kelsey, "Cryptographic support for secure logs on untrusted machines." in *USENIX Security*, 1998.

[12] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine, "A general model for authenticated data structures," *Algorithmica*, vol. 39, no. 1, pp. 21–41, 2004.

[13] A. Haeberlen, P. Kouznetsov, and P. Druschel, "PeerReview: Practical accountability for distributed systems," in *ACM SIGOPS operating systems review*, vol. 41, no. 6. ACM, 2007, pp. 175–188.

[14] S. A. Crosby and D. S. Wallach, "Efficient data structures for tamper-evident logging." in *USENIX Security Symposium*, 2009, pp. 317–334.

[15] L. F. Sarmenta, M. Van Dijk, C. W. O'Donnell, J. Rhodes, and S. Devadas, "Virtual monotonic counters and count-limited objects using a tpm without a trusted os," in *Proceedings of the first ACM workshop on Scalable trusted computing*. ACM, 2006, pp. 27–42.

[16] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: An execution infrastructure for tcb minimization," in *ACM SIGOPS Operating Systems Review*, vol. 42, no. 4. ACM, 2008, pp. 315–328.

[17] R. Kotla, T. Rodeheffer, I. Roy, P. Stuedi, and B. Wester, "Pasture: Secure offline data access using commodity trusted hardware," in *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, 2012, pp. 321–334.

[18] "Trusted Computing Group. TPM v1.2 specification changes." 2003. [Online]. Available: https://www.trustedcomputinggroup.org/groups/tpm/TPM_1_2_Changes_final.pdf

[19] V. Karande, E. Bauman, Z. Lin, and L. Khan, "Sgx-log: Securing system logs with sgx," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 2017, pp. 19–30.

[20] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," *ACM Transactions on Computer Systems (TOCS)*, vol. 33, no. 3, p. 8, 2015.

[21] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. Stillwell *et al.*, "Scone: Secure linux containers with intel sgx." in *OSDI*, 2016, pp. 689–703.

[22] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-sgx: A practical library os for unmodified applications on sgx," in *2017 USENIX Annual Technical Conference (USENIX ATC)*, 2017.

[23] S. Shinde, D. Le Tien, S. Tople, and P. Saxena, "Panoply: Low-tcb linux applications with sgx enclaves," in *Proceedings of the 2017 Network and Distributed System Security Symposium*, 2017.

[24] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "Vc3: Trustworthy data analytics in the cloud using sgx," in *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 38–54.

[25] B. A. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov, "Iron: Functional encryption using intel sgx."

[26] V. Costan and S. Devadas, "Intel sgx explained," Cryptology ePrint Archive, Report 2016/086. http://eprint.iacr.org, Tech. Rep., 2016.

[27] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 640–656.

[28] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring fine-grained control flow inside sgx enclaves with branch shadowing," *arXiv preprint arXiv:1611.06952*, 2016.

[29] N. Weichbrodt, A. Kurmus, P. Pietzuch, and R. Kapitza, "Asyncshock: Exploiting synchronisation bugs in intel sgx enclaves," in *European Symposium on Research in Computer Security*. Springer, 2016, pp. 440–457.

[30] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[31] J. Park, R. Ivanov, J. Weimer, M. Pajic, and I. Lee, "Sensor attack detection in the presence of transient faults," in *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*. ACM, 2015, pp. 1–10.

[32] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[33] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, and S. Capkun, "Rote: Rollback protection for trusted execution." 2017.

[34] K. R. Joshi, M. A. Hiltunen, W. H. Sanders, and R. D. Schlichting, "Probabilistic model-driven recovery in distributed systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 6, pp. 913–928, 2011.

[35] X. Dai and Z. Gao, "From model, signal to knowledge: A data-driven perspective of fault detection and diagnosis," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2226–2238, 2013.

[36] M. A. Alsheikh, S. Lin, D. Niyato, and H.-P. Tan, "Machine learning in wireless sensor networks: Algorithms, strategies, and applications," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1996–2018, 2014.

[37] T. Dierks, "The transport layer security (tls) protocol version 1.2," 2008.