IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 12, 2020, accepted April 20, 2020, date of publication April 30, 2020, date of current version May 14, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2991431

Fine-Grained Access Control-Enabled Logging Method on ARM TrustZone

SEUNGHO LEE[1], HYO JIN JO[2], WONSUK CHOI[3], HYOSEUNG KIM[1], JONG HWAN PARK[4], AND DONG HOON LEE[1], (Member, IEEE)

[1]Graduate School of Information Security, Korea University, Seoul 02841, South Korea
[2]Department of Software Convergence, Hallym University, Chuncheon 24252, South Korea
[3]Division of IT Convergence Engineering, Hansung University, Seoul 02876, South Korea
[4]Department of Computer Science, Sangmyung University, Seoul 03016, South Korea

Corresponding author: Dong Hoon Lee (donghlee@korea.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government (MSIT) under Grant NRF-2017R1A2B3009643.

ABSTRACT Most applications for the Internet of Things operate on embedded systems. In particular, embedded devices intended for smart healthcare, smart homes, and smart cars generate logs containing sensitive user information. These logs must be protected from malicious users while also being accessible for legitimate users to utilize them for providing customized services. Unfortunately, the existing logging system only supporting one-to-one encryption based on a server-client model, so there are limitations in building a decentralized logging infrastructure for the hyper-connected era. In this paper, we propose a new secure logging method that supports one-to-many encryption and extends existing logging systems to a decentralized logging infrastructure. In the proposed method, log publishers are able to encrypt generated logs and distribute them to cloud storage in real time and can ensure that only authorized log subscribers access the logs. For one-to-many encryption, we apply a key-policy attribute-based encryption scheme which is suitable for logging systems. For reliability and efficiency of logs, we apply a key-derivation process that cooperates with one-way hash functions within a trusted execution environment. In a real time logging scenario, the proposed method is 93% faster and occupies 83% less storage space than when an original attribute-based encryption scheme is applied. In addition, performance-tunable parameters can optimize our method for various environments.

INDEX TERMS Embedded system, secure logging, privacy, access controls.

I. INTRODUCTION

The Internet, which started from the Advanced Research Project Agency Network (ARPANET) developed in 1969, has become an indispensable technology [1] and is continuously evolving toward a hyper-connected era in which numerous heterogeneous devices are interconnected. With the development of Internet, advances in the Internet of Things (IoT) technology result in the production of myriad information from interacting with machines, devices, and cars without human intervention. In the future, this information is likely to be utilized by a number of service providers to create customized services such as smart healthcare, smart shopping, smart homes, smart car care, et cetera.

In such a hyper-connected era, a logging system is one of the essential modules for building secure and reliable IoT environments. In general, logs are produced by the logging system according to a de facto standard [2], each of which presents a wide variety of information on events occurring within systems, applications, and networks in a small number of bytes, and as human-readable messages without platform dependency. These logs generally record events in a system. Therefore, they are mainly used in provenance analysis to quickly discover the root cause or ramification of attack symptoms when malicious behaviors are detected in a system [3]–[7]. Recently, cloud service companies present a publish-and-subscribe logging model, i.e., Logging as a Service (LaaS), which can store logs produced by multiple devices to the cloud [8], [9].

This publish-and-subscribe logging model could be applied into various IoT applications. For example, a self-driving car owned by a private individual may act as a log publisher and transmit logs about the vehicle's internal state to a cloud. In this case, a ''subscriber'' denotes a mechanic, insurer, or vehicle manufacturer. The subscribers, therefore, can monitor the status of the vehicle from the transmitted

The associate editor coordinating the review of this manuscript and approving it for publication was Liang-Bi Chen.

logs and link it to marketing activities for customized services.

However, behind the bright side of the hyper-connected era, there is a privacy concern to be addressed. According to [10], one of the major privacy problems of IoT environments is that personal data can be collected without awareness of users. For example, in [11], the careless integration of health data logs into social networks can cause a privacy leakage by identifying Fitbit tracker's vulnerabilities. Like health data, in-vehicle sensor data logs on the controller area network (CAN) can be used to infer who is driving because those sensor data logs indicate driving patterns or habits of a driver [12].

Unfortunately, there are limitations for existing logging systems to be directly applied into a new type of logging environments like a publish-and-subscribe logging model. The reason is that existing logging systems and infrastructures only focus on centralized log management applying one-to-one encryption. It means that logs cannot be utilized by a number of service providers in various places.

In centralized log management, when clients request logs, the server encrypts logs with the key generated via a key exchanged protocol with the clients and transmits the encrypted logs. Furthermore, when multiple clients request logs simultaneously, excessive computational overhead incurs to encrypt logs in addition to key management issues. Therefore, existing logging systems are limited in utilizing encrypted logs in multiple places by only once encryption, and since the server can access all logs, it is difficult to guarantee privacy.

In the case of provenance analysis, the reliability of logs must be guaranteed because the analysis based on manipulated logs leads to incorrect results. Logs inside a system without external access could be reliable, but logs sent to a remote server or cloud storage could be manipulated at unexpected places. Thus, it is difficult to provide sufficient reliability. Therefore, logs used in provenance analysis are mainly used under the assumption that they are reliable.

Therefore, it is desirable for a logging system in the hyper-connected era to be able to guarantee the reliability of logs and provide an infrastructure that allows only authorized log subscriber to access the logs.

In this paper, we propose a fine-grained access control-enabled logging method on a trusted execution environment (TEE), called TEE-aided Log, or T-Log, which addresses the above discussed limitations. T-Log supports fine-grained access control by applying a key-policy attribute-based encryption scheme with minor modification to be suitable for secure logging systems; it provides end-to-end security by encrypting logs with keys derived from one-way hash functions. The key derivation process is advantageous in that a log can be verified directly by deriving a key to match the position of the log, even if previous logs do not exist. The secret keys (i.e., a symmetric key) and the device's unique private-key used by T-Log are handled only within ARM TrustZone, which is widely used in embedded devices. In addition, T-Log is practical, as it works with the plug-in of an existing logging system and provides performance-tunable optimization parameters for various environments.

The main contributions of T-Log are as follows:

- Proposing a secure logging method that supports one-to-many encryption by integrating a key-policy attribute-based encryption scheme and one-way hash functions within ARM TrustZone.
- Implementing the proposed method as a plug-in of an existing logging system to satisfy compatibility and providing performance-tunable parameters to optimize performance in various embedded systems.
- Providing a method to support a *chain of custody* for forensics when adopting logs as legal digital evidence.
- Presenting performance evaluation results and practical overhead based on open platforms.

The remainder of this paper is organized as follows: in Section II, we discuss our proposed T-Log system in the context of related works. In Section III, we provide background knowledge regarding ABE, TEE, and Syslog. In Section IV, we present a system model and threat model. In Section V and VI, we explain the T-Log method and security analysis in detail. Considerations when implementing T-Log and performance results on a real target board can be found in Section VII and Section VIII. In Section IX, we discuss limitations and future research areas regarding our method. Finally, the conclusion is given in Section X.

II. RELATED WORKS

▹**Forward-secure logging:** A study by Bellare and Yee was the first work using one-way hash functions to detect malicious manipulation of logs [13]. The authors reproduced keys to generate a message authentication code (MAC) on every predefined time interval called an ''epoch'' to prevent forged logs by attackers. Schneier and Kelsey proposed a secure logging method generating MAC with previous logs on an untrusted machine to provide manipulation detection of logs [14]. However, the use of one-way hash functions allows detection of replay, replacement, and manipulation of logs, but there are still critical vulnerabilities, in that attackers could exploit keys and information at any time since the keys are used in an untrusted environment.

Karande *et al.* and Paccagnella *et al.* proposed Intel SGX-based logging method which encrypts logs by using symmetric-keys derived from one-way hash functions in Enclave, a commercial TEE solution supported by Intel Software Guard eXtension (Intel-SGX) [15], [16]. However, this approach exhibits limitations extending to a variety of technologies because it is only able to process logs on the same local machine. Recently, researchers have investigated blockchain technology as one way to make logs more reliable. White *et al.* generated immutable logs by using Enclave and sending checkpoints—a digest of logs—to an external blockchain [17]. Since these proposed methods do not support one-to-many encryption, as once encryption, many users

are unable to utilize the encrypted data in various places and do not support efficient access control mechanisms to logs. Thus, it is not sufficient for a decentralized secure logging infrastructure.

▹**IBE and ABE:** ABE is derived from identity-based encryption (IBE). The initial idea for an identity-based cryptosystem appeared in Shamir [18], and Boneh *et al.* subsequently proposed the first IBE able to calculate efficiently based on a pairing operation [19]. Following this, Sahai *et al.* extended IBE to handle various public input during data encryption [20]. ABE is an extension of traditional IBE considering many identities—so-called attributes—rather than a single identity. ABE is classified into ciphertext-policy ABE (CP-ABE) and key-policy ABE (KP-ABE) depending on where the policy composed of access control structure is located. CP-ABE is a policy located on a ciphertext, so it is mainly used to set access permissions for ciphertext when encrypting sensitive data such as personal health records [21], [22]. Therefore, CP-ABE must consider access permissions for a subject who wants to access data before it is encrypting. Recently, [23] proposed a traceable CP-ABE scheme with accountability to address both user and authorization center key abuse.

On the contrary, in the case of KP-ABE, the access policy is located on a secret key and attributes are associated with a ciphertext. Goyal *et al.* proposed a modified KP-ABE scheme that adds a fine-grained access control structure which has demonstrated that ABE can be used to audit a log application [24]. Recently, Zeutro released OpenABE, an open library implemented by C++ that emphasizes access control policies as a cornerstone of these ABE-related studies [25]. However, there is still a revocation problem in ABE. Zeutro explained in the study that this can be solved by expire-date attribute, but the downside is that the size of the key and ciphertext would increase linearly in relation to the number of attributes. Although the ABE scheme is suitable for secure logging systems, it is not efficient for embedded systems. This is because the length of the key and ciphertext increase linearly in proportion to a number of attributes, and it also consumes heavy computing power for pairing operations.

▹**Data and log:** In a hyper-connected era, there is a need for a hybrid method that uses data and logs simultaneously as required. Lee *et al.* surveyed the limitations of motor vehicle event data recorders (MVEDR) [26] to record vehicular data in the event of a car accident and proposed a real-time data recording system called T-Box [27]. However, when T-Box continuously recorded real-time data generated by a vehicle, there was significant network traffic overhead when transferring the recorded data to a destination in addition to a lack of storage space. As a result, it was necessary to separate and manage large amounts of data which contains the current state of a vehicle, and logs which contain short human-readable messages that record events on a vehicle. Lee *et al.* proposed a logging method based on ARM TrustZone that could encrypt and transmit logs without changing the existing Syslog [28], but it made key management difficult between log publishers and subscribers due to the use of only symmetric keys and its inability to utilize logs in various places.

▹**Secrecy:** Previous studies use terminology about secrecy such as *forward secrecy* and *backward secrecy*, but these terms are interpreted differently in group-key agreement protocols and server-client models. As such, this paper uses *past-key secrecy* and *future-key secrecy* suggested by Alzaid *et al.* in [29]. *Past-key secrecy* denotes that the previous key cannot be known by the currently exposed key, whereas *future-key secrecy* indicates that the future-key cannot be known by the current exposed key.

III. BACKGROUND

A. PAIRING-BASED CRYPTOGRAPHY AND KP-ABE

1) BILINEAR MAPS

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order p and let g be a generator of $\mathbb{G}$, then a map $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a (symmetric) bilinear map if it has the following properties.

- Bilinearity: $\forall u, v \in \mathbb{G}$, and $\forall a, b \in \mathbb{Z}_p$, it holds that $\hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}$.
- Non-degeneracy: $\hat{e}(g, g) \neq 1$.
- Efficiency: $\hat{e}(g, g)$ is efficiently computable.

2) COMPUTATIONAL BILINEAR DIFFIE-HELLMAN ASSUMPTION

Let $(p, \mathbb{G}, \mathbb{G}_T, \hat{e}, g)$ be a bilinear group. Given the tuple (g^a, g^b, g^c) where $a, b, c \in \mathbb{Z}_p$, the computational Bilinear Diffie-Hellman (BDH) problem is to compute $\hat{e}(g, g)^{abc}$. The computational BDH assumption holds that solving the problem is infeasible in polynomial time.

3) KEY-POLICY ATTRIBUTE-BASED ENCRYPTION

KP-ABE consists of the following four algorithms:

- $ABE_{Setup}(1^\lambda) \rightarrow (PPs, MK)$. An authority runs this algorithm to generate a master key MK and public parameters PPs associated with MK under security parameter λ.
- $ABE_{KeyGen}(\mathbb{A}, MK) \rightarrow DK$. A key generation center (possibly the same as the authority) having MK runs this algorithm to issue a user's decryption key DK according to access policy $\mathbb{A}$.
- $ABE_{Enc}(PPs, \alpha, \mathcal{M}) \rightarrow CT$. A user who wants to encrypt a message $\mathcal{M}$ with a set of attributes α runs this algorithm. The algorithm outputs ciphertext CT corresponding to α.
- $ABE_{Dec}(CT, DK) \rightarrow \mathcal{M}/\perp$. A user runs this algorithm to decrypt a ciphertext CT using decrypt key DK associated with policy $\mathbb{A}$. If the attribute set of ciphertext α satisfies policy $\mathbb{A}$ (denoted by $\mathbb{A}(\alpha) = 1$), then this algorithm outputs a correct message $\mathcal{M}$; otherwise $\perp$.

To prove KP-ABE, a selective attribute set model under chosen plaintext attacks (CPA) [24] was introduced by

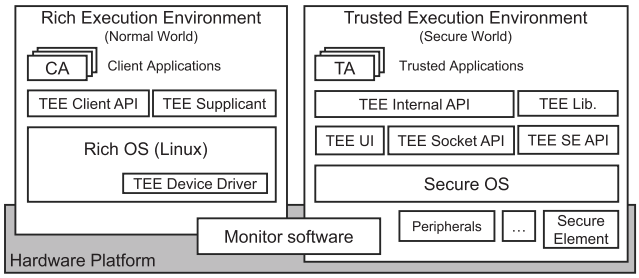


FIGURE 1. ARM TrustZone consists of two worlds according to the non-secure bit, and each world provides libraries and utilities for interaction [30], [31].

defining a game in which a challenger $\mathcal{C}$ interacts with an adversary $\mathcal{A}$ as follows.

$G^{SS-CPA}_{\mathcal{A},ABE}$: Selective-Set CPA game.

- *Init*. $\mathcal{A}$ provides the challenge attribute set α^* to $\mathcal{C}$.
- *Setup*. $\mathcal{C}$ gives *PPs* to $\mathcal{A}$, where *PPs* represents public parameters generated by the ABE setup algorithm.
- *Phase 1*. $\mathcal{A}$ submits an access policy $\mathbb{A}$ where $\mathbb{A}(\alpha^*) \neq 1$ and then receives the corresponding DK from $\mathcal{C}$.
- *Challenge*. $\mathcal{A}$ submits two equal length messages $\mathcal{M}_0$ and $\mathcal{M}_1$. Then $\mathcal{C}$ chooses bit b, and encrypts $\mathcal{M}_b$ with α^*. The challenge ciphertext CT^* is given to $\mathcal{A}$.
- *Phase 2*. Identical to Phase 1.
- *Guess*. $\mathcal{A}$ outputs b' and wins if $b = b'$.

Definition: KP-ABE has CPA security in the case that probability $\mathcal{A}$ wins the game $G^{SS-CPA}_{\mathcal{A},ABE}$ is negligible.

The model for chosen ciphertext attacks (CCA), denoted by $G^{SS-CCA}_{\mathcal{A},ABE}$, can be formalized by accepting the decrypt query without the challenge ciphertext in Phase 1 and 2.

Definition: KP-ABE has CCA security in the case that probability $\mathcal{A}$ wins the game $G^{SS-CCA}_{\mathcal{A},ABE}$ is negligible.

B. TRUSTED EXECUTION ENVIRONMENT

The trusted execution environment (TEE) is a hardware-based isolation technology that provides two separate execution environments that physically share all hardware resources on one processor. TEE provides a more practical trusted computing base for running secure software without wasting hardware resources. Representative commercial TEE solutions are ARM TrustZone and Intel-SGX. Depending on which TEE is chosen, there are different considerations from designing security services to implementation, deployment, and maintenance. This paper focuses solely on ARM TrustZone, which is used for a variety of embedded devices such as mobile phones, IoT devices, vehicles, and industrial systems.

FIGURE 1 shows libraries and utilities for running ARM TrustZone [32]–[34]. ARM TrustZone consists of a normal world, called a rich execution environment (REE), running a Rich OS such as Linux or Windows, and a secure world running a Secure OS. An application running in REE is called a client application (CA), and an application running in TEE is called a trusted application (TA). The TA is a passive application that is called from the CA and provides security services.

Two elements are necessary to operate ARM TrustZone: ARM trusted firmware (ATF) [30], which is responsible for initializing a device and managing boot-chain, and an open portable TEE (OP-TEE) [31], which operates a secure world. ATF provides a secure boot scheme that guarantees a maliciously modified binary will not work until the user program is run from initially run at power up. This only runs properly signed binaries, which prevents an attacker from gaining privilege at boot time. OP-TEE consists of a Secure OS to run TA, related libraries, a TEE service daemon called tee-supplicant that runs as a background process in REE, and a Linux device driver. The tee-supplicant provides a file system and network resource to TEE. The device driver manages memory and communication channels between CA and TA. Related application programming interfaces (API) are standardized by GlobalPlatform [35]. Typical standards include Client API for requesting security service from TA, Internal API for key generation and data encryption inside TA, Socket API for connecting to a remote server with the assistance of tee-supplicant, and SE API for storing sensitive data to a secure element (SE) with tamper-resistant hardware.

Normal worlds and secure worlds are distinguished by a non-secure (NS) bit in the secure configuration register (SCR). Setting the NS-bit is possible only in monitor software running in monitor mode. Switching between normal and secure worlds is generally called world switching. NS-bit in SCR is directly linked with advanced extensible interface (AXI), a main bus line inside system on chip (SoC), and can affect the operating environment of all intellectual property (IP) connected to AXI as a secure or non-secure execution environment [33].

C. SYSLOG

Syslog was developed by researcher Eric Allman, who worked on the Sendmail open source project in 1980, which, since its introduction into Berkeley Software Distribution (BSD), has become a *"de facto standard"* widely used in a variety of unix-based systems. Syslog has evolved continuously since becoming a standard through request for comments (RFC) by Internet engineering task force (IETF).

According to RFC5424 [2], Syslog messages consist of message length, header, and message. Among them, the header field consists of the log priority, Syslog protocol version, log creation timestamp, hostname of the machine that generates the log, application name, and process id, the source and destination IP address and so on. The priority value can be expressed in 8 bits as a numerical code. It has a facility value in a range from 0 (kernel message) to 23 (local7) depending on log use purpose and severity value from 0 (emergency) to 7 (debug). The priority value is calculated by first multiplying the facility value by 8 and then adding the numerical value of the severity.

Representative solutions that improve the stability and scalability of the existing Syslog standard are Rocket-fast

FIGURE 2. Logging infrastructures consist of publishers who generate logs, subscribers who want to utilize logs, and a cloud.

Syslog (Rsyslog) [36] and Syslog next generation (Syslog-ng) [37]. Since 2009, Debian's code name Leny has adopted Rsyslog as the default logger, which is compatible with the existing Syslog configuration and takes into account message processing capacity, processing speed, and scalability [38]. Ubuntu, mainly used by developers, is based on the Debian distribution.

Rsyslog has a modular structure that allows for the adding of various modules as a plug-in for supporting scalability. Rsyslog consists of three modules, input modules for supporting various log input methods, modification modules for reprocessing input messages into desired form, and output modules for storing processed logs as files and databases or for transmitting them to a remote server. Rsyslog's configuration file written in a scripting language can set rules of filtering logs and actions to handle events. Through this configuration file, Rsyslog handles log-flow from input to output modules.

In this paper, we use the log priority to provide access control to logs and a plug-in that is compatible with existing logging systems to support a secure logging mechanism.

IV. OVERVIEW

A. SYSTEM MODEL

Logs are human-readable messages generated by software running inside a device. FIGURE 2 shows an abstraction of logging infrastructure in which publishers produce logs and subscribers utilize logs to provide customized services. In this paper, the publisher is a subject that generates logs and distributes them to a cloud, and the subscriber is a subject that reprocesses and analyzes distributed logs in the cloud.

B. ADVERSARY MODEL

In a logging infrastructure supporting the publish-and-subscribe model as shown in FIGURE 2, the generated logs could be accessed by an adversary such as unauthorized subscribers or a cloud server because logs generated within a device might contain privacy-sensitive information the adversary are interested in. In addition, the adversary can maliciously manipulate logs to cause confusion for authorized subscribers and obstruct the provision of correct customized services. However, we assume that the adversary cannot access to secret values within TEE, e.g., ARM TrustZone; the side-channel attacks on TEE [39] is out of scope of this work.

C. REQUIREMENTS

In the above described logging system, publisher-generated logs must satisfy the following security requirements.

▹**R1. Fine-grained access control**: Logs generated by smart home, smart healthcare, and smart car devices include sensitive information related to user privacy. In a hyper-connected era, a number of subscribers who provide customized services must be able to legally acquire and utilize these logs. In the existing server-client model, one-to-one encryption is applied. It means that existing logging systems only provide log confidentiality through a communication channel, so the server is able to access all logs. This results in a privacy issue over logs. In addition, when a number of subscribers request logs, the server requires tremendous overhead for enencryption and transmission according to the number of requesting parties. To solve this issue, the logging system must provide a one-to-many encryption scheme that can be decrypted in various places with one encryption. Also, fine-grained access control should be applied to the logs based on specific attributes from logs. This requirement can transform existing, centralized log management into decentralized logging infrastructure. This requirement is also included in the ''Data Subject Rights'' of the European Union General Data Protection Regulation (GDPR), which went into effect in May 2018 [40].

▹**R2. Reliability and key protection**: Since logs record various events generated in a device, they can be used as digital evidence for forensics or in provenance analyses to uncover root causes that trigger malicious behaviors in a system. Misjudgments made by using abnormally manipulated logs in log analyses can lead to unexpected results and waste considerable time. For example, vehicular manufacturers may tamper with logs produced from a vehicle involved in an accident to hide significant faults in a vehicle itself, and malicious users could send abnormally modified logs to force the server into poor decision-making. As such, reliability, integrity, authentication, non-repudiation, and immutability of logs are essential security properties. Integrity can detect manipulation of logs, authentication and non-repudiation can verify whether the logs were generated from the correct device, and immutability can preserve the order of logs. Furthermore, to prevent forging, all the relevant keys should be handled within trusted execution environment without any key exposure.

▹**R3. Practicality**: Abandoning the existing logging systems used for decades and introducing a new one to build a decentralized infrastructure is quite inefficient and expensive. Emerging logging systems, therefore, must be compatible with existing systems to guarantee easy application and use. In addition, even if hardware resources are limited such as embedded systems, the logging systems must be able to be effectively applied to various environments. Therefore, optimization parameters must be provided to handle processing overhead in various environments.

V. DESIGN

A. MAIN CONSIDERATIONS FOR DESIGNING T-LOG

▹**Fine-grained access control to logs:** We considered ABE to provide fine-grained access control. As in Section II, ABE

is categorized into KP-ABE and CP-ABE according to where the access structure comprising the policy is located. In this paper, we adopt KP-ABE. Logs are generated by various processes from system-process to user-process. When CP-ABE is applied to the logging system, devices must consider the access structure for each log entry. At this time, it is difficult to predetermine which logs will be required by which subscribers. Furthermore, if the device encrypts logs with a wrong access structure, all logs must be re-encrypted.

By applying KP-ABE, T-Log can quickly encrypt logs without considering the access structure by using specific log message fields as attributes. After that, when a subscriber requests a decryption key, a trusted operator can issue a key, which includes access structure regarding the policy. Thus, applying KP-ABE as a secure logging system is practical. It is also advantageous in that the ciphertext size is proportional to the log message size since only the predefined attributes were used.

Although hashing can provide log immutability, it cannot support fine-grained access control for various subscribers who want to utilize encrypted logs. Therefore, T-Log encrypts logs using keys derived from an initial key via a one-way hash function and encrypts the initial key with KP-ABE to provide access control.

More precisely, we adopt a KP-ABE scheme by Goyal *et al.* [24] and Zeutro [25] with a minor modification. This modification leads to ciphertexts reduced in size and enhances performance for embedded systems. The construction of our scheme is described as follows.

- $ABE_{Setup}(1^{\lambda})$. Under given security parameter λ, this algorithm first defines the universe $\mathcal{U} = \{1, 2., \ldots, n\}$. Then, it randomly chooses $t_1, \ldots, t_{|\mathcal{U}|}, y \in \mathbb{Z}_p$ and generates public parameter PPs and master key MK as shown below:

$$PPs = (T_1 = g^{t_1}, \ldots, T_{|\mathcal{U}|} = g^{t_{|\mathcal{U}|}}, Y = \hat{e}(g, g)^y),$$
$$MK = (t_1, \ldots, t_{|\mathcal{U}|}, y).$$

- $ABE_{KeyGen}(\mathbb{A}, MK)$. The algorithm uses access structure $\mathbb{A}$ and MK. Here, $\mathbb{A}$ is realized by a linear secret sharing scheme (LSSS) [41]. From LSSS, the algorithm can determine random shares λ_i of secret y according to $\mathbb{A}$, where each share is assigned to the corresponding attribute (we omit a concrete description of LSSS and instead refer readers to the study [41]). A decryption key DK is a set of DK_i as shown below:

$$DK_i = g^{\lambda_i / t_i} \text{ for } i \in \{1, \ldots, \ell\},$$

where ℓ is the number of rows in the matrix associated with $\mathbb{A}$ in LSSS.

- $ABE_{Enc}(PPs, \alpha, \mathcal{M})$. The algorithm uses PPs, attribute set α, and message $\mathcal{M} \in \{0, 1\}^k$. Then it chooses a random $s \in \mathbb{Z}_p$. The ciphertext is:

$$CT = (\alpha, C' = \mathcal{M} \oplus H(Y^s), C_i = T_i^s)_{i \in \alpha},$$

where $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ is a hash function.

- $ABE_{Dec}(CT, DK)$. The algorithm uses CT under α and DK associated with $\mathbb{A}$. If $\mathbb{A}(\alpha) \neq 1$ then it outputs $\perp$. Otherwise, it can obtain a set of coefficients $\Omega = \{w_i\}_{i \in \alpha}$ such that $\sum_{i \in \alpha} w_i \cdot \lambda_i = y$ in the LSSS reconstruction manner. Then, for $i \in \alpha$ it computes the following values:

$$Y_i = \hat{e}(C_i, DK_i) = \hat{e}(g, g)^{\lambda_i s}.$$

It finally obtains $\mathcal{M}$ by $C' \oplus H(Y^s)$, $Y^s = \prod_{i \in \alpha'} Y_i^{w_i}$.

▹**Key secrecy and periodic time $\mathcal{T}$:** T-Log satisfies not only past-key secrecy by applying a hash-chain based on a one-way hash function, but also future-key secrecy by updating periodic key $Pkey$ every period $\mathcal{T}$. The initial block key $BKey$ derived from $PKey$ is encrypted with ABE and distributed to the cloud. At this time, if an adversary were to obtain the $BKey$ decrypted with $\mathcal{P}$ computing power, he/she would only be able to decrypt logs processed within $\mathcal{T}$ using the obtained $BKey$, but not the logs produced after time $\mathcal{T}$. If the adversary wished to access these logs, he/she has would be required to spend $\mathcal{P}$ computing power again.

Notably, we use $\mathcal{M}$ as an initial key $BKey_{(0)}$. More precisely, $PKey$ and $BKey$ are derived as follows:

$$PKey_{(t)} = \begin{cases} H(SKey), & \text{if } t = 0 \\ H(PKey_{(t-1)}), & \text{otherwise,} \end{cases}$$

where $SKey$ is a secret key that is shared with stakeholders through the hierarchical shared secret scheme (HSSS), and t is a value of a monotonic counter located in TEE, which is generated by time $\mathcal{T}$.

$$BKey_{(b)} = \begin{cases} H(PKey_{(t)} \mid Tag), & \text{if } b = 0 \\ H(BKey_{(b-1)}), & \text{otherwise,} \end{cases}$$

where Tag is a priority value of a log entry, and b is an index of a block that manages a predefined number of log entries. In addition,

▹**Key protection:** T-Log applies a key-derivation process using a one-way hash function to provide key secrecy and log immutability. This means that if an adversary obtained an initial key, they could easily reproduce all relevant keys. Therefore, T-Log adopts ARM TrustZone as an optimal solution to prevent key exposure in embedded systems. All relevant keys used by T-Log are only handled within ARM TrustZone without key exposure.

B. ARCHITECTURE

T-Log infrastructure requires publishers, subscribers, and a trusted operator (TO) as shown in FIGURE 3. The main role of the TO is to manage access permissions and keys regarding to logs. The TO manages the ruleset and policies related to access permissions set by publishers, and synchronizes with the subscriber T-Log when necessary. When a subscriber requests a key from the TO in order to decrypt specific logs, the TO checks the rules set by publishers regarding the logs and composes a policy accordingly to issue the key. At this time, the policy builds up an access control structure by

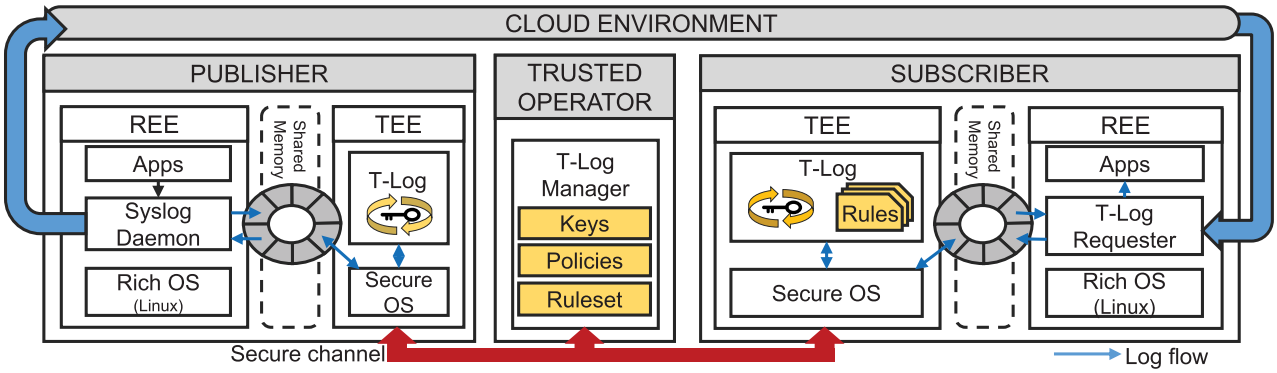


FIGURE 3. T-Log functions across publishers and subscribers. publishers encrypt logs through T-Log and distribute them to various channels. subscribers can decrypt the logs with a decryption key received from the trusted-operator. T-Log and the trusted-operator are synchronized over a secure channel.

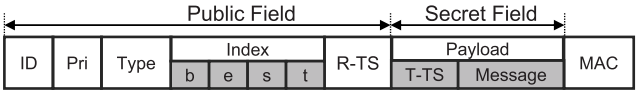


FIGURE 4. An entry is roughly composed of a public-field accessible to everyone, a secret-field to be protected, and a MAC.

using a publisher ID and a log priority managed by Syslog protocol. This makes subscribers only able to access logs that match a policy in *DK* having the publisher's ID and log priority. Details are given in Subsection V-D and Subsection V-E. Publishers, subscribers, and a TO communicate with each other over a secure channel established by the TEE Socket API for rule synchronization. REE and TEE use shared memory to encrypt and decrypt logs. The shared memory uses a ring-buffer for efficient use of memory.

C. ENTRY STRUCTURE

The output of T-Log is generated with a predefined structure called entry. The entry is classified into one of three types. An ABE entry (AE) includes a *BKey* encrypted via ABE and a log entry (LE) includes an original log message encrypted via AES-CBC256. Lastly, the signature entry (SE) contains a signature generated by the device's private key. Each entry must be a type of string presented in ASCII codes for compatibility with Syslog. For this, T-Log encodes each entry as Base64 [42]. This operation maximizes the advantages of the existing Syslog. For example, an entry produced by T-Log can be displayed more efficiently through a console, connected to various output modules configured in a configuration file, transferred to a remote server, or saved as a database through structured query language (SQL).

FIGURE 4 shows an entry structure generated by T-Log. An entry consists of a public-field (PF), a secret-field (SF), and a message authentication code (MAC). PF is not encrypted as a field accessible to everyone, but it should be included when generating a MAC since it affects the integrity of the whole message. Internally, it contains an owner ID, message type, priority, and sequential index. PF is used when a log is later decrypted by subscribers. The ID and priority are used as attributes of ABE while the generated entries are managed by entry units (Index: e). If an entry size is larger than the Syslog message size, it is divided into sub-entries (Index: s) with sub-entry total (Index: t). Rsyslog essentially sets a log message size to 1KB and T-Log fixes the default entry size to 512Bytes, considering the base64 encoding. When the size of an input message is n bytes, the output size is $\lceil \frac{4n}{3} \rceil$bytes. Of course, this setting can be modified depending on the log message size of Rsyslog. When the number of entries included in a block (Index: b) is equal to the predetermined number of entries, a publisher creates a signature for the block and sends it as an entry SE to support non-repudiation. SF consists of log messages sent from Syslogd and a TEE timestamp (T-TS) at the time of T-Log processing, and be encrypted with AES-CBC mode for confidentiality. Finally, the MAC is generated through HMAC-SHA256 with PF and SF.

D. LOG PUBLISHER

A publisher consists of an application that generates logs, a Syslog daemon (Syslogd) that runs as a background process to handle logs, and T-Log. The application sends log messages to Syslogd. More precisely, the application calls the Syslog function in a standard C library (libc). Syslog function is handled in Linux kernel through system call, and Linux kernel delivers application-generated log messages to the Syslogd message queue through Syslog system call handler. This section describes how the T-Log located in the publisher works to provide security properties.

1) FORENSICS-ENABLED KEY DERIVATION

FIGURE 5 shows the entire key derivation process in which the initial block key (*BKey*) is used for one block. T-Log generates a secret key (*SKey*) through a random number generator (RNG) or a password-based key derivation function (PBKDF) to prevent brute force attacks during setup. The generated *SKey* is divided into secrets for participants through hierarchical secret sharing schemes (HSSS), encrypted with

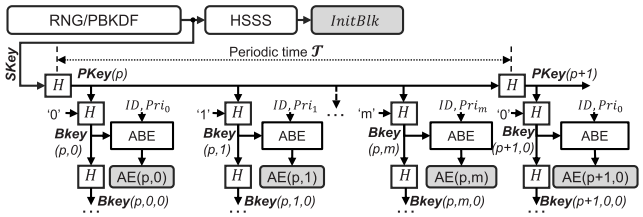


FIGURE 5. T-Log distributes a first-generated *SKey* to each participant through a secret sharing scheme that supports digital forensics and derives *PKey* from *SKey*.

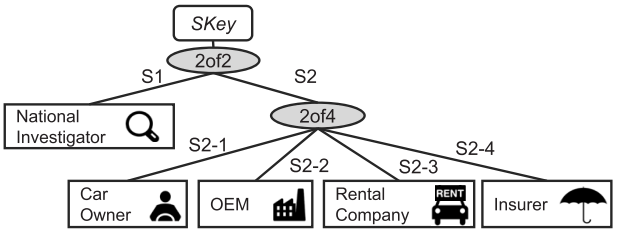


FIGURE 6. Rectangles represent participants, and each ellipse represents the threshold level of secret sharing to reconstruct the *SKey*. *"k of n"* means that if more than *k* secrets are combined from *n* participants, the secret can be reconstructed.

the public-key included in each participant's certificate, and stored in an *InitBlk* (Initial Block). In addition, T-Log creates a signature based on RSASSA-PKCS1-v1.5 2048-bits to support the non-repudiation security requirement for every block unit. The *InitBlk* contains not only secrets but also an extra header-field to involve the time when the T-Log was initialized and the device's serial number. Secrets divided by HSSS can become a restored *SKey* only when there are a number of secrets that meet the set threshold. The restored *SKey* can derive *PKey* and *BKey* through the initialization time of *InitBlk* in the header-field recorded by T-Log. Thus, the *InitBlk* is constructed in a simple as follows: $[Header|RSA_{ENC}(secret_i, PubKey_i)]_{i\in participants}$.

This is for using the log produced by T-Log as forensic digital evidence. In digital forensics, electronic evidence must adhere to a *chain of custody*. This requires chronological documentation from initial data collection to its official use as evidence [43]. If logs generated by T-Log need to be used as legal evidence, HSSS can obtain explicit approval from relevant participants and document when the *SKey* is restored. The keys used to produce an entry can be re-derived only through the reconstructed *SKey*, *InitBlk*'s initial time, and the log's creation time, such that the reliability of the logs can be guaranteed by verifying entries using corresponding keys.

For example, in the case of logs generated from a vehicle, *SKey* can be first divided into two shared secrets $S1$ and $S2$ through HSSS as shown in FIGURE 6. In order to decrypt logs in the event of a vehicle accident in this example, a neutral national investigation agency managing the *chain of custody* must participate, and the threshold level accordingly is set to *"2of2"*. The other secret, $S2$, is composed of four participants related to the vehicle. The threshold is set to *"2of4"* so that at least two participants are required to agree to restore $S2$. This is to consider situations in which one driver does not survive the car accident. As such, HSSS has the advantage of being able to set an appropriate reconstruction condition for each application environment while delivering one secret to all participants [44].

The *PKey* is initially derived from *SKey* and reproduced by hash function every period $\mathcal{T}$ as shown in FIGURE 5. This is to ensure the validity of the key only within that predefined period, even if a strong attacker were to take over a $BKey(p, 0)$. This means that, in past studies, only the *past-key secrecy* for each key was provided through the hash function, but T-Log provides *future-key secrecy* by reproducing the *PKey* every period $\mathcal{T}$.

T-Log processes logs in each branch classified based on the log's priority. The log priority consists of eight bits, so T-Log has a maximum number of 255 branches. When the current time reaches the predefined time $\mathcal{T}$, T-Log signs the digests of the logs processed in each branch and produces an SE. At this time, T-Log derives *BKey* to handle a newly delivered log from *PKey*. If there is an existing block in a branch and the block is full of entries, *BKey* is derived from the previous *BKey*. In this way, T-Log ensures continuity of keys through a key-derivation rule from *SKey* to *BKey*. The generated *BKey* is encrypted with ABE to become an AE and is included in one entry in a block. The *BKey* and the keys needed to encrypt logs are described in the next section.

2) ENCRYPTING A *BKey* BY ABE AND SET ATTRIBUTES

T-Log uses KP-ABE. Thus, publishers can encrypt data quickly by assigning only the attributes according to log contents, without consideration for log access permissions. In FIGURE 5, *BKey* is the first key used in a log block and is encrypted with ABE. The original KP-ABE uses hybrid public-key encryption using asymmetric encryption based on the pairing operation and symmetric encryption based on AES to improve computation performance. Schemes using this key encapsulation mechanism (KEM) usually choose a random number as message $\mathcal{M}$, then encrypts the $\mathcal{M}$ with an asymmetric cipher. After that, it encrypts real input plaintext with a symmetric cipher by using $\mathcal{M}$ as a symmetric key. The Celia library [45] referenced in T-Log implementation is based on the CP-ABE library [46] and uses the KEM. The function of $ABE_{Enc}(PPs, \alpha, \mathcal{M})$, which is part of libcelia encryption, needs PPs, a set α of attributes and message $\mathcal{M}$ which is chosen at random in $\mathbb{Z}_p$. Then it chooses a random $s \in \mathbb{Z}_p$. The result consists of two ciphertexts. $CT1 \leftarrow (\alpha, C' = \mathcal{M}Y^s, C_i = T_i^s)_{i\in\alpha}$ which encapsulates a symmetric-key under the access structure, and $CT2 \leftarrow AES_{ENC}(plaintext, H(\mathcal{M}))$ which encrypts plaintext with the $H(\mathcal{M})$ as the symmetric-key.

In T-Log, only a 256-bit *BKey* is encrypted with ABE, and logs are encrypted using a symmetric key derived from *BKey*. Therefore, T-Log produces only an AE as CT1 without CT2. Let p be an index of periodic time $\mathcal{T}$ and m be a log priority. The *BKey* as the $\mathcal{M}$ derives from PKey: $BKey(p, m) \leftarrow$

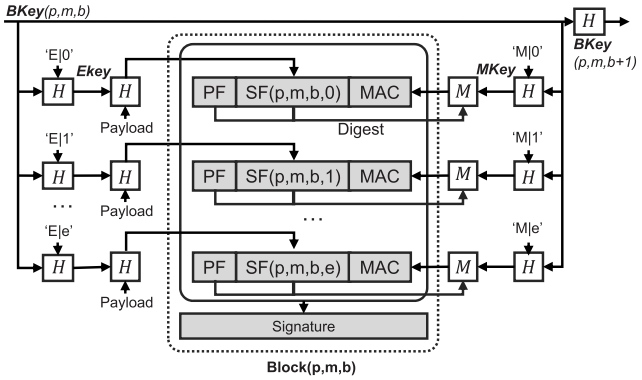


FIGURE 7. T-Log derives an encryption key (*EKey*) and a MAC key (*MKey*) from the current *BKey* to process an arbitrary log and to satisfy the immutability of logs. In order to guarantee the origin of logs, a signature is generated for each block.

$H(PKey(p)|m)_{m\in[0,255]}$. Thus, T-Log produces $CT(p,m) \leftarrow (\alpha, C' = BKey(p,m) \oplus H(Y^s), C_i = T_i^s)_{i\in\alpha}$ and $AE(p,m) \leftarrow [PF|CT(p,m)]$.

T-Log applies two attributes when encrypting a *BKey* to produce an AE, which is the first entry of a block. The first is a publisher ID as a string attribute and the other is a log priority defined in the Syslog protocol as an 8-bit numerical attribute. The priority consists of a combination of facility and severity levels [2]. As the number of attributes in KP-ABE is increasing, the ciphertext size also increases linearly. Therefore, T-Log minimizes the length of ciphertext with policy setting an ID as a string-type attribute and a priority as an 8-bit numerical attribute. The results of the linearly increasing length of keys and ciphertext according to its number of attributes are covered in Section VIII.

3) KEY CHAINING FOR IMMUTABILITY

FIGURE 7 shows the process of encrypting logs and generating a MAC using *EKey* and *MKey* derived from the current *BKey*, which is used to process blocks that are included in a branch classified by a priority, and the *PKey* described earlier is derived from the hash-chain to satisfy the forward integrity property. A similar method was first used in [13] by Bellare and Yee. T-Log derives the keys used in each block from the one-way hash-chain. This way, if there is a modified entry, a subscriber can detect it through MAC verification using a derived *MKey*, and can get assurance regarding entries in a block through signature verification by using a publisher's public-key. In case of MAC generation, applying hash-chain with the previously used key without including a digest of a previous entry, can be directly verified through key derivation without the previous log if there is a request to decrypt an arbitrary log later. The order can also be maintained.

4) EXTENDED TEE INTERNAL API

There are three entry types generated by a publisher as in Section V-C. LE is an entry containing raw logs delivered from Syslogd, and an input log and an output entry are mapped to each other. Thus, it is generated through a synchronized operation between CA and TA. However, an AE and SE should be handled as additional messages through the Syslog function located in REE. For this, T-Log adds a new API called TEE_ReeSyslog in the existing TEE Internal API. The basic framework of TEE_ReeSyslog to support calling REE from TEE has referred to Secure Storage API and TEE Socket API included in TEE Internal API, and a Syslog patch proposed in 2016 [47]. For the extended API, T-Log has modified the optee-os, TEE internal library, and tee-supplicant to extend the interface to the latest OP-TEE. T-Log uses the extended API to pass an AE and LE to tee-supplicant located in REE, and tee-supplicant calls Syslog function to generate additional log messages.

Algorithm 1: The Pseudocode of T-Log Publisher

```
In  : Log /*A log received from Syslogd*/
Out: Entry ∈ {AE, SE, LE}
1  LM ← Log-manager handling global parameters;
2  Create a new entry with PF from Log and LM;
3  Call SyncLogManager();
4  Call KeyDerivation();
5  if message-length > MaxMsgLen then /*split*/
6  |  Split message then set PF.s and PF.t;
7  end
8  Temp ← AES_Enc(SF, EKey);
9  MAC ← HMAC_Gen(PF|Temp, MKey);
10 LE ← [PF|Temp|MAC];
11 TEE_ReeSyslog(LE);
12 LM.branch[pri].digest ←
   H(LM.branch[pri].digest|LE);
13 Increase LM.branch[pri].eid;
14 if LM.branch[pri].eid > NumOfEntry then
15 |  /*send signature*/
16 |  SE ← RSA_sign(LM.branch[pri].digest);
17 |  TEE_ReeSyslog(SE);
18 |  LM.branch[pri].eid ← 0;
19 |  Increase LM.branch[pri].bid;
20 end
```

5) T-LOG ALGORITHM FOR A PUBLISHER

Algorithm 1 to 3 show pseudocode for each sub-modules of a publisher. All logs are handled by the T-Log manager called LM. The publisher receives a log from Syslogd output module and then parses the information needed to construct an entry, such as generated time, priority, log message, and then adds the time processed in T-Log (line 2 in Alg. 1), then it calls SyncLogManager to sync LM with the current time (line 3 in Alg. 1). The added TEE time in a log entry is used to check abnormal situations in which the time variation between the generation time and processing time is large during the verification phase. If the predefined time $\mathcal{T}$ has elapsed, the SE is produced by signing the digest of the log processed so far and delivering it to the tee-supplicant in REE through TEE_ReeSyslog function (line 1-9 in Alg. 2). Next, the *PKey* is reproduced (line 10-15 in Alg. 2), then it calls

Algorithm 2: The Pseudocode of SyncLogManager

```
1  if T is expired then
2      /* check all priority branches */
3      if LM.br[pri].status ≠ empty then
4          /* send signature */
5          SE ← RSA_sign(LM.branch[pri].digest);
6          TEE_ReeSyslog(SE);
7          /* clear branch information */
8          Clear LM.branch[pri];
9      end
10     /* update PKey */
11     sync = [(SF.T-TS − LM.lasttime) / T];
12     while sync ≠ 0 do
13         PKey ← H(PKey);
14         Decrease sync;
15     end
16 end
```

Algorithm 3: The Pseudocode of KeyDerivation

```
1  if PF.e = 0 then
2      /* derive BKey and create AE */
3      if PF.b = 0 then
4          /* first block in priority branch */
5          BKey ← H(PKey);
6      else /* previous block exists */
7          BKey ← H(BKey);
8      end
9      AE ← [PF|ABE_Enc(PPs, α, BKey)];
10     LM.branch[pri].digest ←
       H(LM.branch[pri].digest|AE);
11     TEE_ReeSyslog(AE);
12 end
13 EKey ← H(BKey|"E″|PF.e);
14 MKey ← H(BKey|"M″|PF.e);
```

KeyDerivation to derive keys (line 4 in Alg. 1). If the received log is positioned at the first entry in a branch created based on the log priority, LM derives *BKey* from *PKey*. At this time, if the previous block exists in the branch, it is derived from the previous *BKey* (line 1-8 in Alg. 3). The generated *BKey* is treated as $\mathcal{M}$ in ABE for encryption, and the ciphertext is reconstructed as AE and passed to TEE_ReeSyslog function (line 9-11 in Alg. 3). At this time, $Y^s \in \mathbb{G}_T$ is converted to a fixed 256 bits through element_to_byte API in PBC library and H function, which then encrypts *BKey* with it. Next, the keys for encryption and MAC generation are derived from *BKey* through H function for operation on $\mathbb{Z}_p$ (line 13-14 in Alg. 3). If the length of the message is longer than what Syslogd can handle, it is split into sub-entries and then processed (line 5-13 in Alg. 1). If the number of processed LEs exceeds the number of entries included in the block, an SE from digest for the block is created and transferred to TEE_ReeSyslog function (line 14-20 in Alg. 1).

E. TRUSTED OPERATOR

Trusted operator (TO) is to act as an intermediary between publishers and subscribers along with key-management. TO creates *MK* and *PPs* used in ABE, and delivers *PPs* to the T-Logs located at the publisher and subscriber. The publisher sets the ID and log priority as attributes, then *BKey* is encrypted with the *PPs* to create an AE. The publisher then sets a rule for entries and sends it to the TO. Subscribers can request the TO to issue their *DK*. The TO generates a policy according to the subscriber's request if it is satisfied with the corresponding ruleset and issues generated *DK* to the subscriber or request the publisher to set a rule if the ruleset does not exist. For instance, "$(ID = Alice)$ and $((Pri \geq 8\#8)$ and $(Pri \leq 15\#8))$" states that among entries created by *Alice*, an 8-bit (#8) priority greater than or equal to 8 and less than or equal to 15 can be decrypted, which means the log generated by a user application (Facility: 1) extends from the emergency level (Severity: 0) to the debug level (Severity: 7) according to the Syslog protocol ($Pri = Facility \ll 3 \mid Severity$) in RFC 5424 [2].

F. LOG SUBSCRIBER

As the subscriber T-Log works in reverse to the publisher T-Log in Algorithm 1 to 3, we do not explain in detail. In short, the subscriber provides information about logs that want to decrypt to the T-Log requester (TR). The TR first requests to check the validity of the subscriber *DK* from T-Log. The T-Log validates *DK* via a TO and obtains *BKey* from an associated AE if the *DK* is valid: $BKey \leftarrow ABE_{Dec}(AE, DK)$. Then *EKey* and *MKey* are derived from *BKey* with PF in LE: $EKey \leftarrow H(BKey|\text{“}E''|PF.e)$, $MKey \leftarrow H(BKey|\text{“}M''|PF.e)$. At this time, even though *DK* is valid, if the policy of *DK* issued from the TO does not match the ciphertext attributes, the BKey cannot be decrypted, and the log decryption fails.

If an LE is authenticated by using a derived *MKey*, the SF in the LE is decrypted with *EKey*, and then delivered to TR. Finally, TR delivers the decrypted log to a subscriber. Thus, TR plays a role in assisting T-Log's stable operation. TR can transfer entries stored in local storage or on a database to T-Log, or it can download requested entries from a cloud. In Section VIII, we use entries stored as local files for evaluation.

TO and T-Log operate on the basis of mutual trust and the actual keys are not delivered to users. This mode of operation solves the revocation problem of ABE. Of course, T-Log can solve the revocation problem by using the an expired date attribute proposed by Bethencourt *et al.* in [21], but this means a 32-bit numerical attribute. A large numerical attribute is inefficient because it increases the size of the key and ciphertext, and is not suitable to T-Log, which transfers logs in real time.

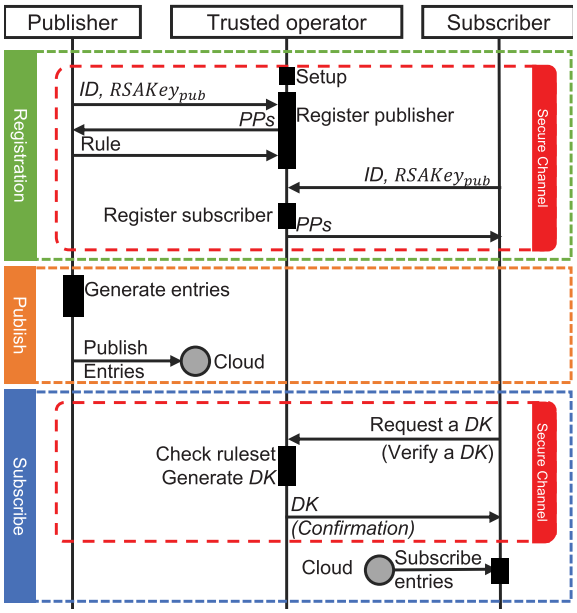


FIGURE 8. The protocol consists of registration, producing, and consuming phases, and sensitive parameters are transferred through a secure channel.

We decided that both publishers and subscribers could resolve this problem if they operate within TEE. Accordingly, T-Log applies minimum attributes to reduce the size of the key and ciphertext, and can control the expiration date of the subscriber with the corresponding ruleset located in TEE. Of course, a disadvantage of this mode of operation is that a subscriber also needs an additional external device. However, service providers can easily accept the additional cost for more reliable log acquisition since the goal is to provide a more suitable service to customers. In addition, T-Log can apply Intel-SGX to subscribers for utilizing logs [48].

G. PROTOCOL

FIGURE 8 shows a high-level protocol between publishers, subscribers, and a TO. It is roughly categorized into registration phases, producing phases, and consuming phases. In the registration phase, a publisher and subscriber generate a public key pair for a digital signature and register the $RSAKey_{pub}$ with a unique ID assigned to each device related to the TO. The TO delivers public parameters PPs generates in ABE_{Setup} to each device in advance. In the producing phase, a publisher encrypts the generated logs with PPs and attributes, then distributes them to the cloud. In order to build a more reliable environment, T-Log can send a digest for each section of logs sent to the blockchain platform. If a subscriber does not have DK, the subscriber may request TO to issue a key or request validation of a previously issued DK. If the DK is valid, the subscriber can decrypt logs stored in the cloud via DK. When publishers and subscribers communicate with a TO, they establish a channel using TEE Socket API to communicate securely.

TABLE 1. Mechanisms for the secure logging system system.

Mechanisms	Security properties								Requirements		
	ID	AuthN	AuthZ	CNF	INT	NR	IMM	TC	R1	R2	R3
KP-ABE	✓	✓	✓	✓					✓	✓	
AES				✓						✓	
Signature						✓				✓	
MAC		✓			✓					✓	
Hash function					✓		✓			✓	
Timestamp							✓			✓	
Index							✓			✓	
ARM TrustZone								✓		✓	
Rsyslog											✓
Tunable parameters											✓

VI. ANALYSIS

A. APPLIED MECHANISMS

TABLE 1 shows the mechanisms applied in T-Log to satisfy the requirements outlined in Section IV-C. Furthermore, it shows the relationship of security properties, such as identification (ID), authentication (AuthN), authorization (AuthZ), confidentiality (CNF), integrity (INT), non-repudiation (NR), immutability (IMM), and trusted computing (TC), provided by the mechanisms in T-Log.

KP-ABE provides various security properties. The ID attribute applied when encrypting logs with KP-ABE can identify publishers, and a trusted operator can authenticate subscribers and issues decryption keys regarding the policy. It only allows authorized subscribers to decrypt the logs. T-Log has applied a minor modified scheme from KP-ABE in Section V-A. The modified scheme in T-Log encrypts BKey, which is an initial key used for a log block managing multiple logs. The proposed scheme is effective in encrypting and transmitting each log generated in real time on embedded systems. We evaluate the performance of T-Log in Section VIII. Even though the KP-ABE scheme that was used generally follows the design for a variant ABE scheme regarding key-encapsulation mechanism [25], the security therein has not been concretely proven as dependent on the scheme. As such, the security guarantee of the KP-ABE scheme used in T-Log is formally described in Section VI-B (R1).

In order to provide log reliability, integrity must be guaranteed to prevent manipulation by malicious users, and the encrypted logs stored in cloud storage must only be decrypted by subscribers who satisfy the policy. In addition, the creation time and sequence of events occurring in the system are very important. In particular, immutability must be guaranteed in digital forensics and provenance analysis. To achieve these properties, T-Log applies HMAC256 as the message authentication code and RSASSA-PKCS1-v1.5 2048-bits as the signature to ensure that the logs are correctly generated by honest publishers. Additionally, T-Log applies AES256-CBC for fast log encryption in embedded systems. To ensure the log immutability, T-Log uses keys derived from a key derivation process using a one-way hash function. At this time, T-Log adds a monotonic counter value and timestamp to the logs, and it handles various keys to satisfy these properties. To ensure that these keys are not exposed, all keys are handled only within a trusted execution environment called ARM TrustZone (R2).

To facilitate compatibility, T-Log operates with a plug-in of Rsyslog, which is an existing logging system in Debian Linux. Furthermore, T-Log provides tunable parameters to optimize performance in embedded systems. Thus, T-Log can be applied to various embedded systems by setting the number of log entries to be included in one block unit or changing the period of time $\mathcal{T}$ (R3).

B. SECURITY GUARANTEE OF KP-ABE IN T-LOG

The security of the original KP-ABE scheme [24] can be proven under the decisional bilinear Diffie-Hellman (BDH) problem [49] by showing that if an adversary enables to break the KP-ABE scheme then the problem could be solved. Following the original strategy, it is possible to show that the aforementioned T-ABE has CPA security. The security proof holds using the computational BDH assumption which is weaker than the decisional one, but it requires the random oracle model [50]. More precisely, consider the following theorem:

Theorem: Let H be modeled as a random oracle. Then the KP-ABE scheme described in Section V-A has CPA security if the computational BDH assumption holds.

Proof: Let $(p, \mathbb{G}, \mathbb{G}_T, \hat{e}, g)$ be a bilinear group and let (g^a, g^b, g^c) be an instance of the computational BDH problem. Then a challenger $\mathcal{C}$ can solve the problem if an adversary $\mathcal{A}$ wins the game $G^{SS-CPA}_{\mathcal{A},ABE}$. $\mathcal{C}$ simulates the game as follows.

- *Init.* $\mathcal{A}$ gives a challenge set α^* to $\mathcal{C}$. In addition, $\mathcal{C}$ controls the hash oracle for H, in which for a query about x, $\mathcal{C}$ outputs g^γ by randomly choosing $\gamma \in \mathbb{Z}_p$ and stores (x, g^γ) in a hash list. If x have been queried before then it outputs the corresponding g^γ from the list.
- *Setup.* $\mathcal{C}$ first sets $T_i = g^{r_i}$ for $att_i \in \alpha^*$ and $T_i = (g^b)^{r_i}$ for $att_i \notin \alpha^*$ where $r_i \in \mathbb{Z}_p$ is randomly chosen, and sets $Y = \hat{e}(g^a, g^b)$. Some elements of MK are implicitly set to unknown exponents regarding either a or b.
- *Phase 1.* For a key query about $\mathbb{A}$ such that $\mathbb{A}(\alpha^*) = 0$, let $M \in \mathbb{Z}^{\ell \times k}$ be the matrix associated with $\mathbb{A}$ in LSSS and let M_{α^*} be the sub-matrix of M consisting of rows assigned by α^*. $\mathcal{C}$ first finds a vector $\vec{u} = (u_1, \ldots, u_k) \in \mathbb{Z}_p^k$ such that $u_1 = 1$ and $M_{\alpha^*}\vec{u} = 0$. These u_is can be efficiently obtained from linear algebra and the relevant proposition is defined in the appendix of the study [24]. Consider a vector $\vec{v} = (v_1, \ldots, v_k) \in \mathbb{Z}_p^k$ randomly chosen by $\mathcal{C}$ and a vector $\vec{s} = b\vec{v} + (ab - bv_1) \cdot \vec{u}$ implying $s_1 = ab$. Recall that the secret shares exist in LSSS and in particular, they are defined to $\lambda_i = M_i\vec{s} \in \mathbb{Z}_p^\ell$ where M_i are the i-th row of M associated with $\mathbb{A}$. Now, the secret key for $\mathbb{A}$ is as follows. For a share assigned to an attribute $att_i \in \alpha^*$, the corresponding secret is $DK_i = g^{M_i\vec{s}/r_i} = (g^b)^{M_i\vec{v}/r_i}$ since $M_i\vec{u} = 0$. Otherwise (i.e., for $att_i \notin \alpha^*$), the secret is $DK_i = g^{M_i\vec{s}/br_i} = g^{M_i(\vec{v}-v_1\vec{u})/r_i}(g^a)^{M_i\vec{u}}$. Obviously, these secrets can be computed with the values that $\mathcal{C}$ knows. Finally $\mathcal{C}$ gives the DK_is to $\mathcal{A}$ as the response.
- *Challenge.* $\mathcal{C}$ simulates a challenge ciphertext of α^* as $CT^* = (\alpha^*, R, (g^c)^{r_i})_{i\in\alpha^*}$ where $R \in \{0,1\}^k$ is a random string and the random value used to encrypt s is implicitly set to the unknown exponent c.
- *Phase 2.* The simulation is same as in Phase 1.
- *Guess.* $\mathcal{A}$ outputs b'.

If $\mathcal{A}$ wins the game (i.e., it tries to decrypt CT^* correctly), $\mathcal{A}$ should ask the hash oracle for $H(Y^s) = H(\hat{e}(g,g)^{abc})$ under the simulation setting. This implies that $\mathcal{C}$ is able to determine $Y^s = \hat{e}(g,g)^{abc}$ from the hash list with probability $1/q_h$ where q_h is the number of the hash queries. Therefore, $\mathcal{C}$ finds the solution of the computational BDH problem with non-negligible probability. The proof is complete. □

Note that the CPA security for KP-ABE can be lifted to CCA security by using one-time signatures. This way was presented along with the original KP-ABE scheme and basically follows the arguments by Canetti, Halevi, and Katz [51]. Consequently, the used KP-ABE scheme having CPA security can achieve CCA security.

VII. IMPLEMENTATION

A. DEVELOPMENT ENVIRONMENT

To develop a trusted application for providing secure services based on ARM TrustZone, T-Log should be implemented on open-platform boards supported by OP-TEE [52]. The boards supporting OP-TEE are able to operate from ROM code that operates at power-up to the required software responsible for trusted services such as ARM Trusted Firmware [30] and Secure OS [31], and Linux [53] maintained by the Linaro security working group.

Among possible development boards, T-Log has selected Raspberry Pi3 Model B (RPi3), which is used in various open communities. In the case of RPi3, TEE platform was activated by Sequitur Labs [54]. It does not have functional problems, but commercial usage is not recommended since not all security features have been verified. The original OP-TEE works on a root file system called Busybox [55], which is one binary including only the essential features as a small executable. Busybox also provides Syslog but does not meet the performance or scalability required by T-Log.

T-Log should communicate with a plug-in of Rsyslog. As explained in Section III-C, Debian applies Rsyslog as the default logging system. T-Log adopts Debian's root file system (RFS) for practicality and compatibility. To do this, T-Log builds a buster version with Debian RFS using debootstrap, a tool that helps build RFS [56].

Note that RPi3 supports symmetric multiprocessing (SMP) with the ARM cortex-a53 quad-core, but only the primary core works when OP-TEE is running. This issue has resolved by modifying the core-enable method to a power state coordination interface (PSCI) in a device tree source (DTS) of a manufacturer [57].

B. MIGRATION OF KP-ABE TO TEE

Zeutro has released an ABE open-source library called OpenABE in 2018 [25]. Unfortunately, the OpenABE is based

TABLE 2. Lines of migrated library code into T-Log.

Category	KP-ABE [45]	PBC [59]	GMP [61]	Extra	Total
Original	2,458	24,811	120,166	-	147,435
T-Log	2,591	6,635	32,941	106	42,273

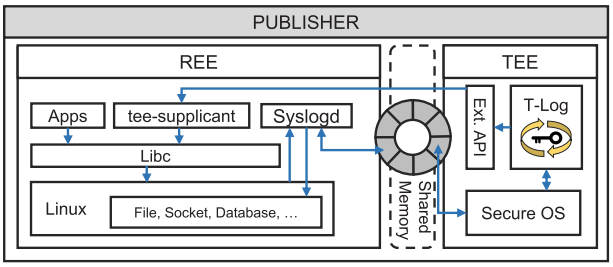


FIGURE 9. T-Log produces entries from received logs from the output module of Rsyslog, then transfers them through shared memory and extended API.

on C++ language, which makes it difficult to apply to OP-TEE, as the current OP-TEE does not support the C++ runtime libraries [58]. To use KP-ABE, T-Log uses Celia library (libcelia) implemented based on C language for access control of electronic health data records [45]. The libcelia extends CP-ABE library [21], [46] to KP-ABE. In addition, they include additional libraries such as a paring-based cryptography (PBC) library [59] for pairing operations, GLib to support various data structures, OpenSSL [60] for cryptographic operations, and GNU multiple-precision (GMP) library [61] for bignum.

In order for KP-ABE to migrate to TEE, it is necessary to remove unnecessary parts from the existing code and replace original APIs to TEE internal APIs supported by OP-TEE. However, the APIs provided by OP-TEE are limited. It means that embedded applications built with libraries provided by sysroot in the ARM cross compiler are not guaranteed to operate in TEE supported by OP-TEE. Therefore, TA must be built on a software development kit (SDK) for TA provided by OP-TEE.

TABLE 2 shows lines of code (LoC) used in T-Log. LoC was counted by an open-source line counter called cloc [62]. T-Log has adopted approximately 29% codes from relevant libraries. OpenSSL library for symmetric encryption and hash functions is replaced with TEE Internal API, and the migrated PBC supports only Type-A pairing, which is the fastest. The Extra consists of redesigned codes that support T-Log operations, which is because some critical input and output functions may result in possible vulnerabilities, such as scanf and printf functions, which are not provided by OP-TEE. The Extra does not include implemented T-Log codes. The line of code that implements the secure logging functions of T-Log, except for the migrated library, is approximately 2,000 LoC.

C. INTERACTION WITH RSYSLOG

Direct communication with T-Log is handled by an implemented output module of Syslogd, a background process of Rsyslog running on REE. The messages processed by each module are all controlled through the configuration file of Rsyslog. FIGURE 9 shows the flow of logs. T-Log receives and handles all logs generated from REE through the output module. In particular, AEs and SEs are delivered to Syslogd through TEE_ReeSyslog extended API as described in Section V-D.4. The AEs and SEs need not be encrypted again with T-Log, so they bypass the output module and arrive at other modules according to filtering rules in the configuration file. In this way, LE, AE, and SE can be delivered to output files, databases, or remote servers via Rsyslog's output modules. In addition, Rsyslog has worker threads in each message queue for fast log processing, such that calling T-Log from the output module does not affect other logs.

TABLE 3. The results of performance evaluation according to curve parameters and attribute settings in KP-ABE.

Parameters /Strength (bits)	Output size of ABE (KB)				Execution time (sec)			
	PPs	MK	DK	CT	Setup	Enc.	KeyGen	Dec.
(r:160,q:512)/80	4.0	1.7	1.9	2.4	0.4	0.2	0.2	0.2
(r:224,q:1024)/112	6.7	1.8	3.0	3.7	1.3	0.5	0.5	0.8
(r:256,q:1536)/128	9.4	1.9	4.1	4.9	3.0	1.1	1.0	2.3
(r:384,q:3840)/192	21.5	2.2	9.2	10.6	22.8	7.8	6.5	23.6
Attribute (Type)	Output size of ABE (KB)				Execution time (sec)			
	PPs	MK	DK	CT	Setup	Enc.	KeyGen	Dec.
String	2.1	0.1	0.4	1.3	1.20	0.26	0.16	0.25
8-bits	9.0	1.8	3.7	4.6	2.83	1.02	0.92	2.09
32-bits	31.0	7.2	14.9	15.5	8.10	3.66	3.56	8.41
64-bits	60.2	14.3	29.8	30.2	15.12	7.16	7.07	16.90

VIII. EVALUATION

In this section, we measure basic performance according to parameters and attributes of KP-ABE on RPi3, and compare the time required to process 1,000 logs of 512 bytes and the storage usage. For reference, Rsyslog processes logs asynchronously through message queues, so there is no significant performance overhead. Thus, this chapter does not include measurement results of Rsyslog working alone. We compare the result of processing 1,000 logs from an original KP-ABE and from T-Log. At this point, KP-ABE does not encrypt all logs at once, and the reason for this is to create the same conditions so that T-Log is able to access arbitrary logs. Thus, KP-ABE encrypts 1,000 logs individually. For evaluation, we compare the performance of T-Log and KP-ABE library proposed in [45] in specific scenarios.

T-Log uses Type A pairing provided by PBC library for symmetric pairing. The pairing operation is based on difficulty of discrete logarithm computation. According to guidance by NIST [63], in a type A parameter, (r:160-bits, q:512-bits) supports security strength of 80 bits, (r:224-bits, q:1024-bits) is 112 bits, (r:256-bits, q:1024-bits) is 128 bits, and (r:384-bits, q:3840-bits) corresponds to 192 bits. Other parameters are not practical on RPi3, so they are not included in this evaluation.

The upper part of TABLE 3 shows output size and execution time evaluated when the number of attributes and plaintext size are fixed and parameters are changed. Considering suitable parameters that satisfy both security strength and processing time, (r:256-bits, q:1536-bits) sup-

porting a minimum of 128-bits strength was chosen. For reference, each parameter is generated by the PBC *gena-param* utility. The lower part of TABLE 3 shows the output size and execution time when various attributes are applied using the previously chosen parameters. Attributes can have string or numerical attributes, wherein numerical attributes are represented as "bag of bits" so that a number of attributes generated are proportionate to the bits. Therefore, when using a numerical attribute, it should be used with minimal bits.

To solve the key-revocation issue in ABE, Bethencourt *et al.* proposed using an expired date as a numerical attribute in [21], but when using a 64-bit numerical attribute in an embedded system as shown in TABLE 3, encryption takes approximately 7 seconds and decryption takes approximately 17 seconds. This is not practical for a real-time logging system in embedded systems. Therefore, in order to improve logging performance, T-Log applies a method to validate DK via TO and solve key-revocation issue. Of course, this method has additional communication overhead, but we focused on the logging performance for embedded systems. Thus, T-Log applies two attributes as a string attribute expressing the ID and an 8-bit numerical attribute expressing log priority from 0 to 255. The two attributes produce low performance overhead on RPi3 and are suitable for operating T-Log.

There are two ways to encrypt logs. The first method is to encrypt a log file when a file generated by a certain process is larger than the predefined size and then transmit it to cloud storage, and the second method is to encrypt each log entry and transmit it to cloud storage in real time. When applying the first method, the results are identical with T-Log since KP-ABE also operates with KEM. However, in this case, individual log entries cannot be transmitted in real time and decryption of each log entry becomes impossible. Therefore, we focused on the second method to transmit the log generated in a high-speed network environment in real time. When using KP-ABE, significant encryption time is consumed for each log entry, and the ciphertext size increases due to ABE operation. However, T-Log encrypts only BKey with KP-ABE and, subsequently, generated logs are quickly encrypted with AES using a key derived from the BKey, which can reduce encryption overhead and ciphertext size.

We compare time spent and ciphertext size when 1,000 512-byte logs were operated in two cases of KP-ABE [45] and T-Log. The T-Log period $\mathcal{T}$ is set to 10 seconds and the number of entries in the block is fixed to 16.

In FIGURE 10, the KP-ABE consumed 1.1 seconds to encrypt a single log, producing a total time (TT) of 1,080 seconds. T-Log only took 50 milliseconds to process an LE, 200 milliseconds when including an SE that is created every 16 entries, and AE, SE, and LE together took 1.7 seconds and consumed 77 seconds in total. Compared to encrypting each log with KP-ABE, T-Log reduced the processing time by almost 93%.

FIGURE 11 shows the size of ciphertext as output. KP-ABE generates an approximate ciphertext size (CS) of 5KB per log, resulting in an approximate total ciphertext size (TC)

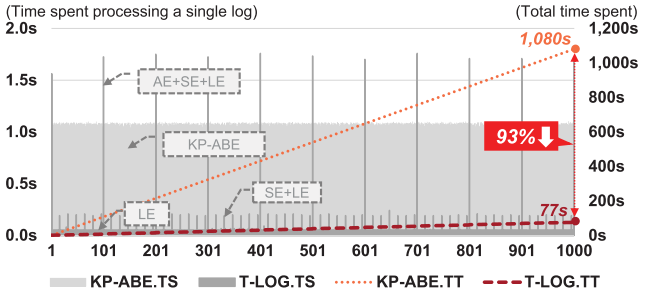


FIGURE 10. Processing time to publish secure logs.

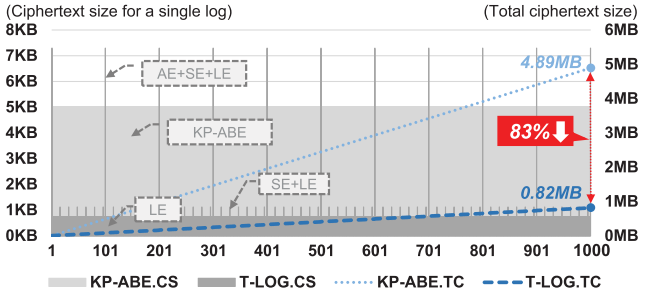


FIGURE 11. Size of ciphertext.

of 4.89 MB. This is almost 10 times that of 1,000 inputs of 512-bytes. In contrast, the size of the total entry encrypted by T-Log was approximately 0.82 MB. The encrypted output logs are 26% larger than the size of the total input log entries, and it includes 33% overhead for each entry due to base64 encoding. As a result, even in base64-encoded results in T-Log, the ciphertext size was reduced by almost 83% compared to KP-ABE. Of course, KP-ABE is more efficient than T-Log when processing logs accumulated over a certain period were stored in a local file. When encrypting 1,000 logs stored in a file, KP-ABE took 1.18 seconds and the ciphertext size was 504 KB. However, applying an original KP-ABE cannot handle an arbitrary entry and is not suitable for processing logs in real time.

IX. DISCUSSION AND FUTURE RESEARCH

A. DISCUSSION

One of the most important features for a logging system is logging performance, or how many logs can be processed per second. Existing logging systems without log encryption take only a few milliseconds to process thousands of logs, and Paccagnella *et al.* presented a secure logging system to record more than one million events per second based on Intel-SGX [16]. However, T-Log operates on embedded devices with limited hardware resources, so it consumes more overhead than other logging systems operating on servers. Thus, it is efficient in selectively encrypting and transmitting only filtered logs by Syslog rules, instead of applying T-Log to all logs generated by a device. In addition, if the key exchange is possible between heterogeneous TEEs such as Intel-SGX and ARM TrustZone, subscribers can use Intel-SGX to decrypt

logs quickly without leaking keys identical to the method proposed in this paper.

T-Log focuses on the study of logging methods to support log reliability and to allow multiple authorized subscribers to use logs in various ways. Although this paper does not cover the authentication of log publishers, the problem can be resolved through a secure boot scheme provided by ATF and SELinux. In short, logs being passed to T-Log can be delivered only through a registered named-pipe, and SELinux can manage all related access subscriber permissions.

Attacks regarding logs can be classified into four types: replay, deletion, modification, and truncation. T-Log can detect replays, deletions, and modifications, but it does not cover the situation where a system goes down along with a truncation attack before signatures are produced block by block. T-Box solved this truncation attack by adding a flag bit, indicating whether it is the last or a subsequent continuing piece of data [27]. However, since the logs produced by T-Log are sent to a cloud environment in real time, truncation attacks are not considered an immediate threat. In addition, when T-Log for subscribers decrypts logs, the scenario of an insider attacker maliciously storing logs in internal storage and delivering the logs through an abnormal channel to an unauthorized subscriber is not considered. We believe that this can be solved via additional functions, such as digital rights management (DRM) with T-Log. For the purposes of this paper, this topic was deemed out of scope and was not discussed in detail.

B. FUTURE RESEARCH

In recent years, edge computing is emerging as a solution for reducing computational overhead in a centralized cloud environment [64], [65]. The proposed method does not support a hierarchical structure for such distributed processing, but applying the ABE-based key delegation method [66] or outsourced encryption and decryption scheme suitable for cloud storage systems [67] can distribute overhead that is concentrated on a trusted operator. Furthermore, if the outsourced ABE scheme can be used in T-Log, a scheme proposed by Li *et al.* in [23], [68], which provides continuous leakage resilience against side-channel attacks by updating decryption keys, would be a good approach to build more secure logging infrastructure.

In addition, T-Log can cooperate with blockchains, such as an Edge IoT framework, integrating a blockchain and internal cryptocurrency system to manage an edge computer resource pool [69]. T-Logs supporting many-to-many relationship between publishers and subscribers can collaborate with smart contracts in a phase of decryption key generation. A subscriber can request the blockchain to issue a decryption key through a smart contract. A trusted operator can obtain the publisher's approval, issue a corresponding key for the event sent through the smart contract, and subsequently pay cryptocurrency to the publisher, who is the owner of the requested logs.

We have seen the possibility of a secure logging system using KP-ABE operating on ARM TrustZone through this paper. We will continue to research more efficient KP-ABE schemes for embedded systems that can cooperate with cloud and convergence technologies.

X. CONCLUSION

In this paper, we first extracted requirements and security properties for secure logging systems and suggested practical mechanisms to satisfy these requirements. Then, we designed and implemented a secure logging method called T-Log that satisfies all security properties. To date, T-Log is the first logging solution to apply KP-ABE within ARM TrustZone where relevant keys can be securely managed. Since our method provides fine-grained access control and end-to-end security, it can make many-to-many relationships between log publishers and subscribers, and will become a cornerstone to creating a number of customized services. In addition, for the practicality of logs, T-Log was implemented as a plug-in of Rsyslog, which is a default logging system in Debian Linux, and demonstrated that it could be easily applied while remaining compatible with existing logging systems without degrading performance. In addition, by providing performance-tunable optimization parameters, i.e. the number of entries included in a block and the period $\mathcal{T}$, performance can be optimized for various environments. Furthermore, T-Log is able to resolve KP-ABE's revocation problems by applying an operational way to validate DK through a secure channel with TO. The described features make T-Log a dependable and secure logging infrastructure for the emerging era of hyperconnectivity.

ACKNOWLEDGMENT

The authors would like to thank the editors and anonymous reviewers who gave them valuable feedback to improve the quality of this article.

REFERENCES

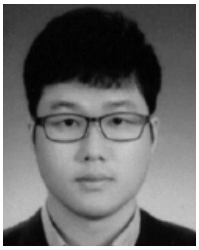
[1] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. B. Postel, L. G. Roberts, and S. S. Wolff, "A brief history of the Internet," *Comput. Commun. Rev.*, vol. 39, no. 5, pp. 22–31, 2009.
[2] R. Gerhards, *RFC 5424: The Syslog Protocol*, document IETF 5424, Request for Comments, 2009, pp. 1–38.
[3] Y. Liu, M. Zhang, D. Li, K. Jee, Z. Li, Z. Wu, J. Rhee, and P. Mittal, "Towards a timely causality analysis for enterprise security," in *Proc. NDSS*, 2018.
[4] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Towards automated log parsing for large-scale log data analysis," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 6, pp. 931–944, Nov. 2018.
[5] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1285–1298.
[6] A. D'Alconzo, I. Drago, A. Morichetta, M. Mellia, and P. Casas, "A survey on big data for network traffic monitoring and analysis," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 3, pp. 800–813, Sep. 2019.
[7] W. U. Hassan, M. A. Noureddine, P. Datta, and A. Bates, "OmegaLog: High-fidelity attack investigation via transparent multi-layer log analysis," in *Proc. NDSS*, 2020.
[8] Splunk. *The Data-to-Everything Platform*. Accessed: Mar. 5, 2020. [Online]. Available: https://www.splunk.com
[9] Elastic. *Elastic Logs*. Accessed: Mar. 5, 2020. [Online]. Available: https://www.elastic.co

[10] R. Chow, "The last mile for IoT privacy," *IEEE Secur. Privacy*, vol. 15, no. 6, pp. 73–76, Nov. 2017.
[11] M. Rahman, B. Carbunar, and M. Banik, "Fit and vulnerable: Attacks and defenses for a healthmonitoring device," *CoRR*, vol. abs/1304.5672, Aug. 2013. [Online]. Available: http://arxiv.org/abs/1304.5672
[12] M. Enev, A. Takakuwa, K. Koscher, and T. Kohno, "Automobile driver fingerprinting," *Proc. Privacy Enhancing Technol.*, vol. 2016, no. 1, pp. 34–50, Jan. 2016.
[13] M. Bellare and B. Yee, "Forward integrity for secure audit logs," Dept. Comput. Sci. Eng., Univ. California San Diego, San Diego, CA, USA, Tech. Rep., 1997, vol. 184.
[14] B. Schneier and J. Kelsey, "Cryptographic support for secure logs on untrusted machines," in *Proc. 7th USENIX Secur. Symp.*, vol. 98, 1998, pp. 53–62.
[15] V. Karande, E. Bauman, Z. Lin, and L. Khan, "SGX-log: Securing system logs with SGX," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Apr. 2017, pp. 19–30.
[16] R. Paccagnella, P. Datta, W. U. Hassan, A. Bates, C. W. Fletcher, A. Miller, and D. Tian, "Custos: Practical tamper-evident auditing of operating systems using trusted execution," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020.
[17] R. White, G. Caiazza, A. Cortesi, Y. I. Cho, and H. I. Christensen, "Black block recorder: Immutable black box logging for robots via blockchain," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3812–3819, Oct. 2019.
[18] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Proc. CRYPTO*, 1984, pp. 47–53.
[19] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," *SIAM J. Comput.*, vol. 32, no. 3, pp. 586–615, Jan. 2003.
[20] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Proc. EUROCRYPT*, 2005, pp. 457–473.
[21] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2007, pp. 321–334.
[22] J. Eom, D. H. Lee, and K. Lee, "Patient-controlled attribute-based encryption for secure electronic health records system," *J. Med. Syst.*, vol. 40, no. 12, pp. 253:1–253:16, Dec. 2016.
[23] J. Li, Y. Zhang, J. Ning, X. Huang, G. S. Poh, and D. Wang, "Attribute based encryption with privacy protection and accountability for CloudIoT," *IEEE Trans. Cloud Comput.*, to be published.
[24] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. 13th ACM Conf. Comput. Commun. Secur. (CCS)*, 2006, pp. 89–98.
[25] *The OpenABE Design Document Version 1.0*, Zeutro, 2018, p. 29.
[26] *IEEE Standard for Motor Vehicle Event Data Recorders (MVEDRs) Amendment 1: MVEDR Connector Lockout Apparatus (MVEDRCLA)*, IEEE Standard 1616-2004, 2004.
[27] S. Lee, W. Choi, H. J. Jo, and D. H. Lee, "T-box: A forensics-enabled trusted automotive data recording method," *IEEE Access*, vol. 7, pp. 49738–49755, 2019.
[28] S. Lee, W. Choi, H. J. Jo, and D. H. Lee, "How to securely record logs based on ARM TrustZone," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Jul. 2019, pp. 664–666.
[29] H. Alzaid, D. Park, J. M. G. Nieto, C. Boyd, and E. Foo, "A forward and backward secure key management in wireless sensor networks for PCS/SCADA," in *Proc. S-CUBE*, 2009, pp. 66–82.
[30] (ARM). *ARM Trusted Firmware GitHub*. Accessed: Mar. 5, 2020. [Online]. Available: https://github.com/ARM-software/arm-trusted-firmware
[31] OP-TEE. *OP-TEE GitHub*. Accessed: Mar. 5, 2020. [Online]. Available: https://github.com/OP-TEE
[32] V. Costan and S. Devadas, "Intel SGX explained," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 86, 2016.
[33] *Cortex-A53 MPCore Processor*, Technical Reference Manual, ARM, Cambridge, U.K., 2014.
[34] *Security Technology Building a Secure System Using Trustzone Technology*, ARM Ltd., Cambridge, U.K., 2009.
[35] GlobalPlatform. *GlobalPlatform*. Accessed: Mar. 5, 2020. [Online]. Available: https://globalplatform.org
[36] Adiscon. *The Rocket-Fast Syslog Server*. Accessed: Mar. 5, 2020. [Online]. Available: https://www.rsyslog.com/
[37] *Syslog-NG. The Foundation of Log Management*. Accessed: Mar. 5, 2020. [Online]. Available: https://www.syslog-ng.com
[38] Debian. *Debian Rsyslog*. Accessed: Mar. 5, 2020. [Online]. Available: https://wiki.debian.org/Rsyslog
[39] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "Armageddon: Cache attacks on mobile devices," in *Proc. USENIX Secur.*, 2016, pp. 549–564.
[40] GDPR. (2018). *GDPR Key Changes*. Accessed: Mar. 5, 2020. [Online]. Available: https://gdpr.eu/tag/gdp
[41] A. Beimel, *Secure Schemes for Secret Sharing Key Distribution*. Haifa, Israel: Technion-Israel Institute of Technology, Faculty of Computer Science, 1996.
[42] S. Josefsson, *RFC 4648: The Base16, Base32, and Base64 Data Encodings*, document RFC 4648, Request for Comments IETF, 2006, pp. 1–18.
[43] E. Casey, *Digital Evidence and Computer Crime—Forensic Science, Computers and the Internet*, 3rd Ed. New York, NY, USA: Academic, 2011.
[44] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.
[45] Y. Zheng, "Privacy-preserving personal health record system using attribute-based encryption," M.S. thesis, Worcester Polytech. Inst., Worcester, MA, USA, 2011.
[46] J. Bethencourt, A. Sahai, and B. Waters. *Ciphertext-Policy Attribute-Based Encryption*. Accessed: Mar. 5, 2020. [Online]. Available: http://acsc.cs.utexas.edu/cpabe
[47] J. Forissier. *Send Traces to Tee-Supplicant*. Accessed: Mar. 5, 2020. [Online]. Available: https://github.com/jforissier/optee_os/tree/syslog
[48] S. Lee, W. Choi, H. J. Jo, and D. H. Lee, "Poster: Secure logging infrastructure employing heterogeneous trusted execution environments," in *Proc. Symp. Netw. Distrib. Syst. Secur. (NDSS)*, 2020.
[49] D. Boneh and M. K. Franklin, "Identity-based encryption from the Weil pairing," in *Advances in Cryptology CRYPTO* (Lecture Notes in Computer Science), vol. 2139, J. Kilian, Ed. Berlin, Germany: Springer, 2001, pp. 213–229.
[50] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proc. ACM Conf. Comput. Commun. Secur. (ACM CCS)*, New York, NY, USA, 1993, pp. 62–73.
[51] R. Canetti, S. Halevi, and J. Katz, "Chosen-ciphertext security from identity-based encryption," in *Advances in Cryptology EUROCRYPT* (Lecture Notes in Computer Science), vol. 3027, C. Cachin and J. Camenisch, Eds. Berlin, Germany: Springer, 2004, pp. 207–222.
[52] Linaro. *Platforms Supported*. Accessed: Mar. 5, 2020. [Online]. Available: https://optee.readthedocs.io
[53] *Linaro Security Working Group Linux GitHub*. Accessed: Mar. 5, 2020. [Online]. Available: https://github.com/linaro-swg/linux
[54] Sequitur. (2016). *Easing Access to ARM TrustZone, OP-TEE and Raspberry Pi3*. Accessed: Mar. 5, 2020. [Online]. Available: https://connect.linaro.org/resources/las16/las16-111
[55] D. Vlasenko. (2012). *BusyBox*. Accessed: Mar. 5, 2020. [Online]. Available: https://busybox.net
[56] Debian. *Debootstrap*. Accessed: Mar. 5, 2020. [Online]. Available: https://wiki.debian.org/Debootstrap
[57] Antonio. *Linux Bootstrap for Raspberry Pi3 With Trusted Firmware-A*. Accessed: Mar. 5, 2020. [Online]. Available: https://github.com/AntonioND/rpi3-arm-tf-bootstrap
[58] ARM. *OP-TEE Frequently Asked Questions*. Accessed: Mar. 5, 2020. [Online]. Available: https://optee.readthedocs.io/en/latest/faq/faq.html
[59] B. Lynn. *The Pairing-Based Cryptography Library*. Accessed: Mar. 5, 2020. [Online]. Available: https://crypto.stanford.edu/pbc
[60] OpenSSL. *The OpenSSL Project*. Accessed: Mar. 5, 2020. [Online]. Available: http://www.openssl.org
[61] GNU. *The GNU Multiple Precision Arithmetic Library*. Accessed: Mar. 5, 2020. [Online]. Available: https://gmplib.org
[62] A. Danial. *Count Lines of Code*. Accessed: Mar. 5, 2020. [Online]. Available: https://github.com/AlDanial/cloc
[63] E. Barker, "NIST special publication 800-57 part 1, revision 4," NIST Special Publication, Gaithersburg, MD, USA, Tech. Rep. 800-57, 2016.
[64] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[65] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017.
[66] S. Kumar, Y. Hu, M. P. Andersen, R. A. Popa, and D. E. Culler, "JEDI: Many-to-many end-to-end encryption and key delegation for IoT," in *Proc. USENIX Secur.*, 2019, pp. 1519–1536.
[67] J. Li, W. Yao, Y. Zhang, H. Qian, and J. Han, "Flexible and fine-grained attribute-based data storage in cloud computing," *IEEE Trans. Services Comput.*, vol. 10, no. 5, pp. 785–796, Sep. 2017.

[68] J. Li, Q. Yu, Y. Zhang, and J. Shen, "Key-policy attribute-based encryption against continual auxiliary input leakage," *Inf. Sci.*, vol. 470, pp. 175–188, Jan. 2019.
[69] J. Pan, J. Wang, A. Hester, I. Alqerm, Y. Liu, and Y. Zhao, "EdgeChain: An edge-IoT framework and prototype based on blockchain and smart contracts," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4719–4732, Jun. 2019.

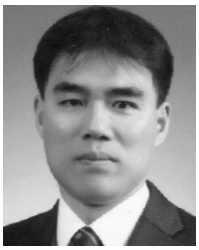
SEUNGHO LEE received the B.S. and M.S. degrees in computer science from Pukyong National University, Busan, South Korea, in 2006 and 2010, respectively. He is currently pursuing the Ph.D. degree in information security with the Graduate School of Information Security, Korea University, Seoul, South Korea. In the past, he has worked on embedded system security projects for the Internet of Things (IoT). Since 2006, he has been working with the Samsung Advanced Institute of Technology, Samsung Electronics Corporation, Yongin, South Korea. His research interests are embedded system security, secure logs, and applied cryptography.

HYO JIN JO received the B.S. degree in industrial engineering and the Ph.D. degree in information security from Korea University, Seoul, South Korea, in 2009 and 2016, respectively. From 2016 to 2018, he was a Postdoctoral Researcher with the Department of Computer and Information Systems, University of Pennsylvania, USA. In 2018, he joined the Department of Software Convergence, Hallym University, Chuncheon, South Korea, as an Assistant Professor. His research interests include security for cyber physical systems, applied cryptography, and privacy-preserving methods.



WONSUK CHOI received the B.S. degree in mathematics from the University of Seoul, Seoul, South Korea, in 2008, and the M.S. and Ph.D. degrees in information security from Korea University, Seoul, in 2013 and 2018, respectively. He was a Postdoctoral Researcher with the Graduate School of Information Security, Korea University. In 2020, he joined the Division of IT Convergence Engineering, Hansung University, Seoul, as an Assistant Professor. His research interests include security for body area networks, usable security, applied cryptography, and smart car security.

HYOSEUNG KIM received the B.S. degree in mathematics from Korea University, Seoul, South Korea, in 2010, where he is currently pursuing the Ph.D. degree in information security with the Graduate School of Information Security. His research interests include functional encryption, functional signature, and anonymous attestation.



JONG HWAN PARK received the B.S. degree from the Department of Mathematics, Korea University, Seoul, South Korea, in 1999, and the M.S. and Ph.D. degrees from the Graduate School of Information Security, Korea University, in 2004 and 2008, respectively. From 2009 to 2011, he was a Research Professor with Kyung Hee University. From 2011 to 2013, he was a Research Professor with Korea University. Since 2013, he has been an Assistant Professor with the Department of Computer Science, Sangmyung University, Seoul. His research interests include functional encryption, broadcast encryption, authenticated encryption, and various cryptographic protocols.

DONG HOON LEE (Member, IEEE) received the B.S. degree from the Department of Economics, Korea University, Seoul, South Korea, in 1985, and the M.S. and Ph.D. degrees in computer science from The University of Oklahoma, USA, in 1988 and 1992, respectively. He is currently a Professor and the Director of the Graduate School of Information Security, Korea University. He has been with the Faculty of Computer Science and Information Security, Korea University, since 1993. His research interests include cryptographic protocol, applied cryptography, functional encryption, software protection, mobile security, vehicle security, and ubiquitous sensor networks (USNs) security.