DistLog: A Distributed Logging Scheme for IoT Forensics

Hassan N. Noura[1], Ola Salman[1], Ali Chehab[1], and Raphaël Couturier[2]

[1]Electrical and Computer Engineering, American University of Beirut (AUB), Beirut, Lebanon
[2]Univ. Bourgogne Franche-Comté (UBFC), FEMTO-ST Institute, CNRS, Belfort, France

***Abstract*—Digital forensics are vital in the Internet of Things (IoT) domain. This is due to the enormous growth of cyber attacks and their widespread use against IoT devices. While IoT forensics do not prevent IoT attacks, they help in reducing their occurrence by tracing their source, tracking their root causes and designing the corresponding countermeasures. However, modern IoT attacks use anti-forensics techniques to destroy or modify any important digital evidence including log files. Anti-forensics techniques complicate the task for forensic investigators in tracking the attack source. Thus, countermeasures are required to defend against anti-forensics techniques. In this paper, we aim at securing the IoT log files to prevent anti-forensics techniques that target the logs' availability and integrity such as wiping and injecting attacks. In the proposed solution, and at regular intervals of time, the logs generated by IoT devices are aggregated, compressed and encrypted. Afterwards, the encrypted logs are fragmented, authenticated and distributed over n storage nodes, based on the proposed Modified Information Dispersal Algorithm (MIDA) that can ensure log files availability with a degree of $(n-t)$. For data dispersal, two cases are considered: the case where the fog nodes are interconnected and the case where they are not. For the former case, the n obtained fragments are transmitted to n neighboring IoT devices (aggregation nodes). However, for the latter one, the output is transmitted to the corresponding fog and then, dispersed over the n neighboring fog nodes. A set of security and performance tests were performed showing the effectiveness and robustness of the proposed solution in thwarting well-known security attacks.**

***Keywords*—** Protecting Log Files; Anti-Forensics; Anti-Forensics IoT; IoT digital forensics; Log files availability; Preserving evidence

I. Introduction

Digital forensics is a branch of forensic science that deals with the investigation of digital crimes, which are basically attacks on digital devices. There are many sub-categories of digital forensics based on the targets, including computer forensics, network forensics, mobile forensics, etc. The ability to carry on digital investigation is directly related to the availability of evidence, mainly within log files. These include banks of data, which can help investigators in retrieving information about the crime's causes and tracing back their sources (computer, laptop, tablet) [1]. This, consequently, leads to the detection of the attacker's location and/or seizure. In the era of the Internet of Things (IoT), digital forensics will play a key role in investigating the crimes committed against IoT devices [2], [3]. However, modern IoT attacks employ anti-forensics techniques to hide any evidence that can be traced back to the attackers. One of these techniques is to delete or modify the log files.

In this context, securing the logs of IoT devices is critical to limit the number and effect of security attacks. To secure these logs, many solutions have been proposed in the literature, primarily using cryptographic solutions that consist of encrypting the logs and saving them at the device level or at a trusted third party. However, the majority of IoT devices exhibit limited processing and storage capacities and thus, securing and keeping the log files at the device level is not practical nor effective.

Log files should be protected in a reliable and robust manner that prevents their modification or loss. Hence, ensuring the integrity, confidentiality, availability and authentication of log files is essential in the IoT forensics domain (or in any network of interconnected devices such as the case of drones and robots).

The importance of IoT log files has been recently highlighted by some attacks, which also targeted the deletion or modification of these logs to complicate the IoT forensics investigation [4]. The aim of this work is to ensure the confidentiality, integrity, source authentication, in addition to the availability of these log files.

The rest of this paper is organized as follows. Section I presents the related work to logs security. In Section III, we give an overview about the main logs-related attacks, secret sharing, compression and aggregation. In Section IV, the proposed solution for securing IoT log files is presented including the network model, and the threat model. Section V describes the derivation process of the required cryptographic primitives needed for the proposed scheme. In Section VI, we present our proposed logs security scheme. Section VII analyzes the security of the proposed scheme against the main log-related attacks. Section VIII discusses the cryptanalysis results including tests for randomness, uniformity and key sensitivity. In Section IX, we evaluate the performance of the proposed scheme in terms of storage and computation overhead. Finally, Section X concludes this article.

II. Related Work

Securing log files has always been a major goal to the area of digital forensics. Several techniques have been proposed to preserve the logs' confidentiality, integrity, or availability.

A. Encryption-based

Encryption-based solutions, to preserve the logs' confidentiality, rely on either symmetric or asymmetric encryption. Symmetric algorithms have the advantage of being less computationally expensive, but with the strict requirement of securing the shared secret key. On the other hand, the asymmetric techniques resolve the problem of key management, but exhibit a higher complexity and require a specialized infrastructure (Public Key Infrastructure (PKI)) for key distribution.

In [5], Bellare and Yee proposed a scheme based on forward integrity; the Message Authentication Codes (MAC) of the received logs are added sequentially, and the symmetric key is updated at regular intervals. As such, if one key is compromised, modifying past entries is not a straightforward task. Although this scheme preserves the logs' integrity, however, it does not ensure their confidentiality and availability. A similar scheme was proposed by Schneier and Kelsey in [6] to secure the logs by a chain of log MACs. Thus, the secret key is pre-shared between the logging devices and updated after each log entry. Waters et al. proposed to secure logs using public key schemes in [7]. Holt et al., in [8], proposed Logcrypt, which combines public-key cryptography with MAC, resulting in digital signatures of the corresponding logs.

Ma and Tsudik proposed FssAgg [9], which stands for "Forward-Secure Sequential Aggregate", a scheme that uses an aggregated chain of signatures to achieve public verification and thwart malicious attacks aiming at deleting logs. This way, any device having the public key is able to verify the integrity of the received log. Extending FssAgg, Yavuz et al. proposed the Blind-Aggregate-Forward (BAF) solution [10], an efficient and publicly verifiable cryptographic scheme. However, the verification of one log entry requires the verification of all log entries. Additionally, if the verification fails, it is not possible to know which entry has been modified. Thus, the authors proposed a variant of Fast-Immutable BAF (FI-BAF) in [11] to solve the problem of fine-grained verification. FI-BAF saves the aggregation of the signatures and the individual ones, leading to a storage overhead double that of BAF. In a more recent work [12], Yavuz et al. proposed the Log Forward-secure and Append-only Signature (LogFAS) scheme to ensure high level verification with storage efficiency, using the Schnorr signature scheme. It consists of verifying the aggregation of logs instead of verifying each log separately. Recently, in [13], Hartung et al. showed that LogFAS [12] and two variants of FssAgg [9] are prone to four attacks enabling secret key recovery and log forgery.

Other than the hash-chain approaches, another work was proposed in [14] based on Merkle trees. In [15], Crosby and Wallach proposed a scheme based on a history tree, which enables the verification of individual log events in an efficient way. Tree-based schemes are storage efficient but they require logarithmic complexity, in terms of computations, to generate integrity proofs.

B. Secure Hardware

Another approach for securing logs is based on hardware security, such as the introduction of security-related instruction codes, the Software Guard Extensions (SGX). SGX-Log was proposed by Karande et al. in [16] to secure the logs by saving them on a single SGX node. In addition to the security provided by cryptographic solutions, SGX ensures the security of the keys. In this case, the logs confidentiality and integrity are preserved but their availability is not. Nguyen et al. proposed Cloud-based secure logger for medical devices in [17]. The scheme consists of streaming medical logs to an Intel SGX-based server application (enclave). The communication of logs is secured by Transport Layer Security (TLS). Also, a hash-chain containing a signature of each record is computed upon the receipt of each log record. Log hash-chains are verified by the application server using the sealing mechanism. In [18], Sinha et al. combined block-based hash-chains with SGX to secure the log integrity and confidentiality.

Nguyen et al. proposed LogSafe to ensure the Confidentiality, Integrity and Availability (CIA) of logs pertaining to IoT devices [19]. LogSafe consists of storing the device logs on one or more cloud nodes running SGX. The confidentiality is ensured by applying the Advanced Encryption Standard (AES) encryption algorithm. The integrity is ensured by using hash chaining. In addition, a counter-based snapshot is used to defend against replay attacks. The availability is ensured by performing backup on a different node at a different location to alleviate the risk of Distributed Denial of Service (DDos) attacks. However, the proposed solution has limited availability protection, given that a backup is done on only one device and presents a storage overhead given that the data is duplicated. EmLog was proposed in [20]. The constrained nature of IoT devices was considered for supporting Trusted Execution Environments (TEEs). The proposed solution stores the logs in a centralized entity based on a dynamic key approach.

C. Cloud and Blockchain Based

Another secure log storage approach was based on the Cloud by introducing the secure storage-as-a-service. In [21], secure logging-as-a-service was proposed by pushing the logs to the cloud. Similarly, a cloud-based log storage scheme was proposed in [22]. More recently, block-chain based storage was considered for a distributed storage solution. In [23], the authors benefit from the distributed nature of blockchains to propose a decentralized storage and verification system for IoT

digital evidence. In the proposed solution, the meta-data and hash are saved in Cyber Blockchain Trust (CBT). However, the privacy of evidence is not well protected given that they are saved in a permission-ed ledger built on top of HyperLedger Fabric. Moreover, the computational challenge associated with the blockchains' proof-of-work make them inappropriate for IoT limited devices.

As summarized in TABLE I, many of the existing solutions exhibit a high overhead in terms of processing and storage requirements, and as such, they are not designed for IoT systems. Moreover, the scalability of these solutions is constrained by the enormous amounts of logs generated within the IoT domain, which makes the chain-based verification unfeasible (e.g.blockchain). On the other hand, ensuring availability by duplicating logs (i.e. backup) presents a high storage overhead. In addition, storing logs on a secure hardware at the cloud does not prevent the availability attacks. Thus, there is a need for a lightweight and efficient scheme that protects logs' confidentiality, integrity and availability. In this context, IDA is a promising solution since it exhibits a small overhead in terms of computations, communication and storage, when compared to other secret sharing variants such as the Shamir scheme. Moreover, IDA requires less storage overhead compared to traditional backup solutions.

The main contributions of this paper stem from the limitations of the existing work, within the IoT forensics domain, for the protection of log files, as summarized below:

- The design of a distributed logs' security scheme based on a lightweight message authentication-encryption algorithm, in addition to the Modified IDA to ensure all data security services, availability, integrity, and confidentiality with source authentication.
- Modifying the IDA scheme to ensure logs' availability based on a distributed storage scheme. The logs are encrypted and then separated into n fragments to be stored at n IoT devices or fog nodes. This guarantees data redundancy since only t out of n fragments are needed to reconstruct the original logs.
- The encryption algorithm and the hash function are lightweight, with minimal resource requirements, yet offering a high security level. The dynamic key-based approach is adopted to generate the permutation, selection, and update tables, in addition to the IDA matrices, which strikes a good balance between the required performance and security level.

III. Background

In this section, we discuss the main attacks that might compromise the security of the logs. Also, we briefly explain the main concepts behind our scheme including aggregation, compression, secret sharing and fog computing.

A. Log Files and IoT Digital Forensics

Every application includes some sensitive log files, which are considered the primary source of information for forensics investigations. As such, these must be secured, especially if located within non-trustworthy devices such as the case of IoT devices. If an attacker compromises an IoT device, the logs should be protected from any manipulation since these are used to identify, analyze and diagnose any occurring event within these devices. In fact, attackers are able to discover and scan for any design flaw, misconfiguration, software bug, or some exploitable vulnerability to compromise these devices, and logs can be used to track the events around the attack, to trace it and investigate its causes.

Due to the vast adoption of IoT technology, the need for security and forensics becomes essential. From a cyber-crime perspective, log files (used for monitoring and accountability) should not be destroyed nor modified to preserve important evidence. In fact, IoT logs' security relies on the existing security measures of the IoT devices (authentication and control access schemes). Any weakness in the implementation of these security measures will lead to exposing these logs to unauthorized alteration [24].

B. Log File Attacks

The IoT device auditing process consists of storing events within a log file. This provides the detection capability of suspicious activities, along with further examination and inspection through the data gathering process. Such a process must be performed periodically, and it can be checked and reviewed either locally, or remotely. For this reason, logs became the target of attackers to cover up their tracks. Attackers rely on the following anti-forensic attacks to modify or delete logs:

1) Log Wiping Attack: Data wiping is a technique used to overwrite the currently available data file(s) with a flow of ones and zeros, or with random characters over all sectors of a corresponding file(s) stored on a digital device. Suck an attack would hinder the forensic investigation, and it can be launched remotely, quickly, and using limited resources.

2) Log Injection Attack: This technique, based on modifying the log files, is also used to cover the tracks and source of any cyber attack. Obviously, this invalidates the log files and render them useless for further investigation.

Moreover, an attacker may also inject malicious codes and scripts by relying on steganography methods [25], or by uploading a malicious program (from an un-trusted malicious source) to elevate their privilege on the compromised device(s) such as computers [26], [27] or smart-phones [28].

C. Fog Computing

This recent technology aims at pushing data services to the user proximity. Being an extension of cloud computing, fog computing allows processing to be performed at the network edge, and thus, enables innovative IoT applications [29], [30], especially those that require very low latency. On the other hand, the edge network will handle the transmission of the large amounts of data generated by IoT networks, while only long-term data will be sent to the cloud. Accordingly, fog and cloud computing will operate in a complementary manner to enhance the overall network performance. Also, fog

Category	Approach	Limitations
Authentication-Encryption Algorithm	• Symmetric Authentication-Encryption Algorithm [5], [6]: This type of solutions can ensure data confidentiality, data integrity and source authentication of log contents. • Digital Signature (Asymmetric) + Symmetric Encryption [7], [9], [10], [12], [13], [9], [14], [15]: This solution can ensure non-repudiation in addition to the previous listed security services.	However, these solutions cannot ensure log availability if end devices are compromised.
Secure Hardware	SGX based [16], [17], [18], [19], [20]	Security requires specialized hardware and data availability is ensured by saving multiple copies of the same data, which introduces high communication and storage overhead.
Cloud based	Logging-as-a-service [21]	Pushing the logs to the cloud does not guarantee their availability unless backup is done.
Blockchain-based	Saving logs hash and metadata [22], [23]	Handling large number of devices would result in unexpected ledger sizes and consequently high computation and resources requirements.

TABLE I: Literature Summary

computing is supposed to play an essential role in providing security services in the IoT domain, such as the identification and authentication of IoT devices. Securing IoT data at the fog level is essential when limited devices are incapable of executing strong encryption algorithms, and the same applies for the security of IoT logs to enable IoT forensics.

D. Aggregation

Data aggregation solutions play a major role in improving the network effectiveness. Its primary objective is to collect and aggregate data packets to decrease the required resources and delay such as power consumption, congestion of traffic, in addition to maximizing the network lifetime. A comparative analysis of several data aggregation schemes for IoT applications was presented in [31]. The authors quantified a set of performance metrics such as energy consumption, network lifetime, throughput, latency and number of nodes alive. Based on this study, they recommended the use of LEACH-C in case of a centralized and selective clustering approach.

E. Compression

Data Compression techniques can either be lossy or loss-less. The difference is that loss-less compression allows the reconstruction of the original data without any degradation, while lossy compression entails a low degradation. However, the advantage of lossy compression is its ability to achieve better data reduction [32], which reduces the communication and storage overhead.

In our case, log contents should be compressed in a loss-less manner, and their integrity should be preserved due to their importance as evidence in digital forensics. A set of loss-less compression techniques is listed in [33]. One of the most efficient techniques is the zStandard (zstd) [34], which achieves efficient compression with a very good performance. Thus, we adopt this technique in our proposed solution.

F. Information Dispersal Algorithm

The Information Dispersal Algorithm (IDA) [35] divides data of size d into t fragments, each of size $l = \frac{|d|}{t}$. Thus, the data is reshaped into a matrix form, DM, with $(t \times l)$ elements. IDA consists of multiplying an $(n \times t)$ non-singular generator matrix (G) by DM to produce an encoded matrix EDM with a size of $(n \times l)$. Any t rows of EDM could be used for the reconstruction of the original data. The IDA encoding step is defined by the following matrix equation:

$$EDM = G \odot DM \tag{1}$$

Where $\odot$ represents the matrix multiplication operation.

The inverse IDA process, to recover the original data, uses matrix multiplication of the inverse matrix of G_t and that of the t received fragments, according to the following equation:

$$DM = G_t^{-1} \odot EDM_t \tag{2}$$

Where G_t consists of t rows of the IDA matrix G and G_t^{-1} is its corresponding inverse. In addition, EDM_t represents any t fragments of EDM.

Information dispersal adds redundancy compared to the original data with minimum storage overhead ($(n - t) \times l$) compared to the original Shamir secret sharing [36]. Note that it is impossible to explicitly reconstruct the original data from a number of fragments less than t.

IV. Proposed IoT Logs Security Scheme

In this section, we present the network model that we consider for securing the logs in addition to the proposed IoT logs' security scheme workflow.

A. Network Model

We consider a network model, as illustrated in Fig.1, which consists of an IoT network with n interconnected devices. Some of these devices can serve as storage devices or aggregation nodes. Each device can be connected to one fog node (or gateway). In addition, the IoT network might be subdivided into multiple IoT networks, where each network is connected to one fog node (or gateway).

The proposed solution consists of applying sequential aggregation, loss-less compression, encryption and authentication processes on the collected logs during a session, then fragmenting and distributing them over n

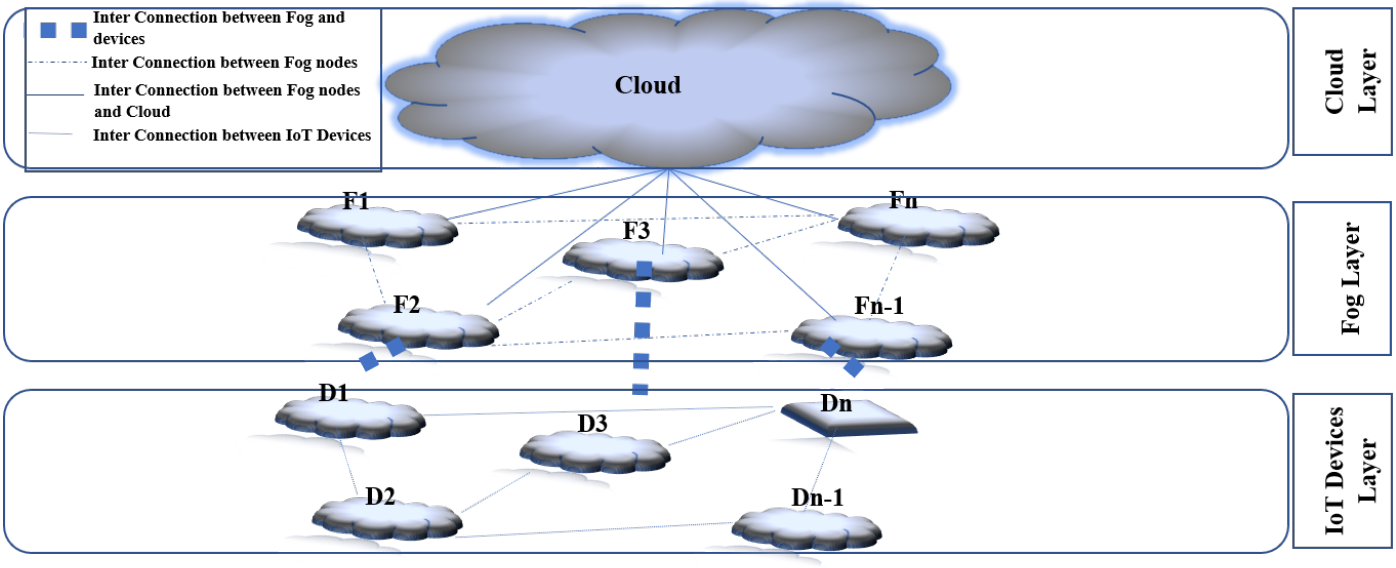


Fig. 1: Network Model

entities. A session is defined as the interval of time during which the device has been authenticated and communicating with the gateway (i.e. fog node). Each time the device is re-authenticated, a new session is defined.

In case the fog nodes are not interconnected, the logs are distributed over n IoT aggregation nodes. In this case, an aggregation node is assumed to be a powerful IoT device (having sufficient computation and storage resources), and as such, the fog node is able to send the fragmented encrypted data to these nodes.

In the other case where the fog nodes are interconnected, each fog node collects logs from its connected IoT devices (or IoT network). Then, the fog node processes the collected logs and disperses them over to the neighboring n fog nodes.

B. Logs Security Scheme Workflow

For both scenarios, only t encrypted fragments are required to recover the original logs. Therefore, the proposed solution for both scenarios consists of six main steps (see Fig. 2), as listed below:

1) Logs normalization, aggregation and compression
2) Dynamic key derivation and construction of cryptographic primitives; this process is performed once per session. The session time depends on the application, the logs type, size, and granularity level. These cryptographic primitives (selection tables and permutation table) are updated for each input log. The update process is based on a permutation operation using specific permutation update tables.
3) Encryption using a dynamic permutation cipher scheme. For each session, this permutation table is updated using its corresponding dynamic update permutation table.
4) Availability process using the proposed Modified Information Dispersal Algorithm (MIDA) (described in detail in Section VI-D); the encrypted logs are reshaped into a matrix with t rows, each corresponding to an original fragment. This matrix is divided into a set of sub-matrices, where each is encoded using one of the m produced IDA matrices.
5) Dispersion process: it is performed by selecting the storage nodes in a pseudo-random manner(n nodes). Also, the distribution of these fragments, among the nodes, is done in a dynamic manner.
6) The choice of t and n depends on the application and the required security level. Increasing t and/or n increases the required execution time, but increases the security level, especially for availability.

An outline of the proposed cryptographic solution is presented in Fig. 7. For consistency, the used notations are listed in Table II.

All the cryptographic operations (cipher, authentication and availability primitives) are based on a dynamic key, which is generated for each new session, by using a master secret key, SK, and a nonce. This dynamic key is divided into a set of sub-keys, which are used to construct the cryptographic primitives.

Note that the inverse of the cryptographic solution is not detailed, given that it consists of the same operations with minor changes, such as the use of the inverse matrices in the multiplication operations of MIDA) and the inverse permutation table to decrypt the data. The inverse process takes place when a security incident takes place. In this case, the authority responsible for investigating the incident requests the logs, at specified time intervals, from the corresponding fog node. The latter, having the corresponding dynamic keys and a map of the storing devices, re-assembles the corresponding t fragments

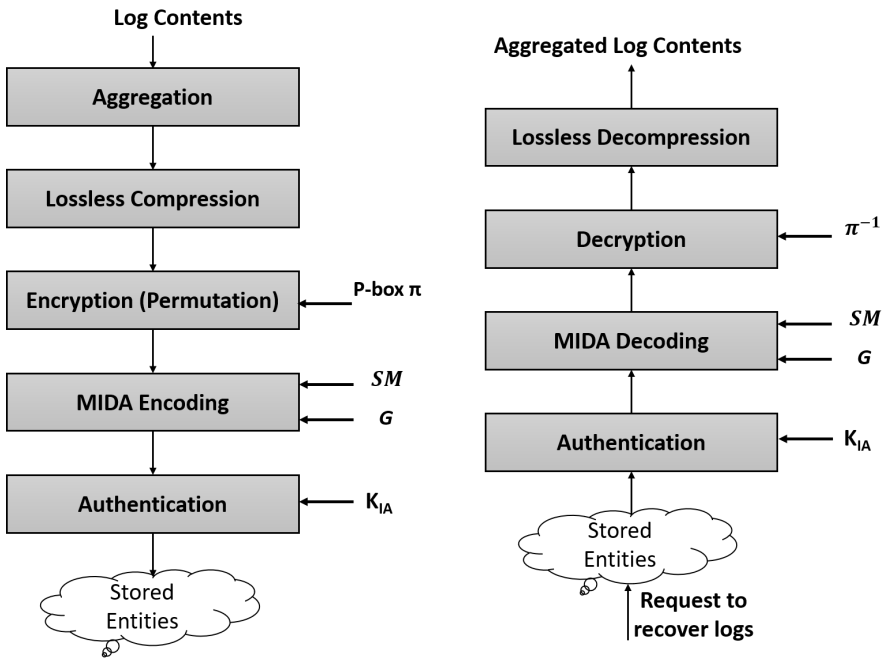


Fig. 2: (a) Proposed logs preserving scheme, (b) and its corresponding inverse solution, respectively

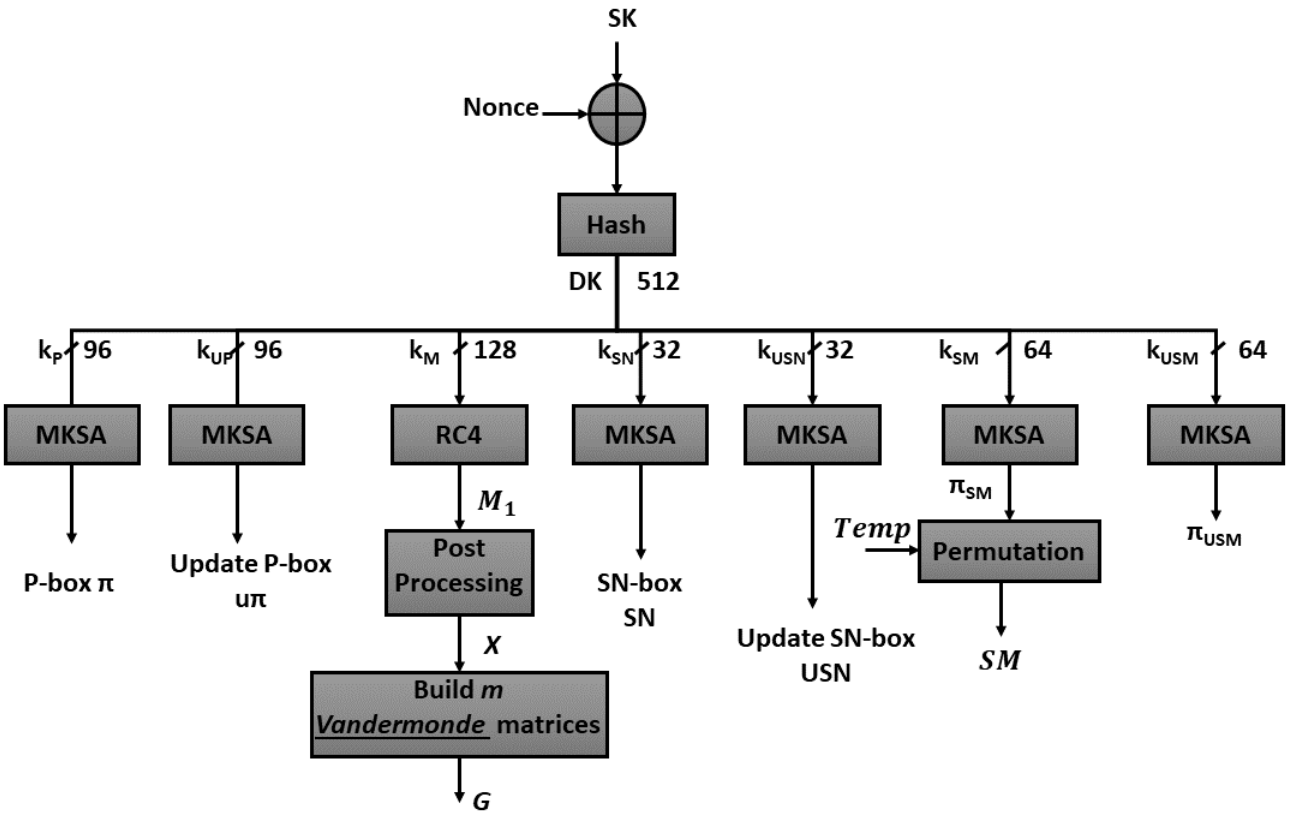


Fig. 3: Proposed key derivation function and its corresponding cipher and update primitives generation process

to reconstruct the requested logs, and checks for integrity and authentication compliance.

V. Construction of Cryptographic Primitives

In this section, we describe the derivation process of the cryptographic primitives needed for the proposed MIDA scheme, more specifically, the proposed dynamic key derivation scheme, the construction of the MIDA matrices, and the permutation and selection tables.

A. Proposed key derivation scheme

The proposed solution is based on the dynamic key approach, and all cryptographic primitives are related to this key. To reduce the computational overhead when generating new keys for each new session, update tables are used for the update. The proposed dynamic key and sub-keys derivation is illustrated in Fig. 3. The process takes the following two inputs:

- **Secret key SK:** This master secret key can be exchanged, depending on the application, using several techniques such as a Key Distribution Center (KDC) or using a Public Key Infrastructure (PKI) [37]. The length of SK can be equal to 128, 192, 256 or 512 bits.
- **Nonce N_o:** It can be generated using a pseudo-random generator at the fog or aggregation node level. A new $Nonce$ will be generated for each new session, and it is associated with the device logs and the corresponding time interval at the fog node. As such, the corresponding fog node is able to retrieve such information for the reverse process.

The secret key, SK, and the nonce, N_o, are xor-ed together and then, the result is hashed to generate the dynamic key, as expressed below:

$$DK = h(SK \oplus N_o) \tag{3}$$

where h stands for a cryptographic hash function and DK for the obtained dynamic key. We use SHA-512 [38] since it exhibits good cryptographic properties and it can resist the different collision attacks. Accordingly, the output size of the dynamic hash function is 64 bytes. Eq. (3) guarantees that a one-bit change in the $Nonce$ results into a completely different dynamic key.

Next, the dynamic key is divided into seven sub-keys; each one is used to construct either a cryptographic primitive or an update for such a primitive. The derivation process of the dynamic key and corresponding sub-keys is illustrated in Fig. 3. For each new session, a new nonce is generated, which leads to the generation of a different dynamic key. This, in turn, results into completely different set of cipher primitives.

More specifically, DK, which consists of 512 bits, is divided into seven sub-keys $\{K_P,\ K_{UP},\ K_M,\ K_{SN},\ K_{USN},\ K_{SM},\ K_{USM}\}$ with the following respective sizes: 96, 96, 128, 32, 32, 64, and 64 bits. Below is a description of the role of each sub-key:

- K_P: it is used to produce a dynamic permutation table that is used to encrypt the log contents. The data encryption process is based on permutation at the byte-level. K_P consists of the most significant 12 bytes of DK.
- K_{UP}: it consists of the next most significant 12 bytes of DK, and it is used to produce an update table for the permutation sub-key.
- K_M: it is used to generate the IDA Matrices. It has a size of 128 bits and it is used as a seed for a stream cipher to construct a key-stream of a total size of $(m \times n)$ bytes. Then, it is divided into m blocks, each with n bytes. Next, we eliminate zeros and repeated values within each block. Finally, the blocks are used to generate the IDA Vandermode matrices.
- K_{SN}: it is used to generate a selection table when selecting either cluster nodes or fog nodes. It consists of the fourth set of least significant 4 bytes of DK.
- K_{USN}: it is used in the update process of the selection table. It consists of the third set of least significant 4 bytes of DK.
- K_{IA}: it is used in the integrity and authentication process. First, we perform the xor of K_P and K_{UP}, and the xor K_{SN} and K_{USN}, and then we concatenate the two results as expressed by the following equation:

$$K_{IA} = K_p \oplus K_{up} || K_{SN} \oplus K_{USN} \tag{4}$$

  The length of K_{IA} is 128 bits and it is sufficient to prevent message authentication attacks.
- K_{SM}: it is used to generate the IDA matrices selection table, SM. It represents the next least significant 8 bytes of DK.
- K_{USM}: it is used in the update process of the selection table SM. It represents the least significant 8 bytes of DK.

B. Construction of Cryptographic Primitives

In this section, we describe the construction techniques of the cryptographic primitives such as the permutation and selection tables, in addition to the IDA matrices (m).

1) Dynamic Encryption Permutation Table: In this work, the modified $KSA-RC4$ algorithm, proposed in [39], is used to produce a dynamic flexible permutation table of variable length, len. According to [39], the obtained permutation table π exhibits robust security properties for a key ≥ 32. Moreover, since the produced permutation table has the bijective property, the inverse permutation table π^{-1} can be computed by using the following equation:

$$\pi^{-1}[\pi(i)] = i,\ i = 1,\ 2, \ldots,\ len \tag{5}$$

where $\pi(i)$ is the i^{th} element of π and len is the length of the produced permutation table.

2) MIDA Matrices: In contrast to the original IDA, which uses one matrix, the proposed MIDA scheme uses m matrices. Each IDA matrix consists of a block with n distinctive and non zero bytes from X. Generating these matrices requires the iteration of a stream cipher, such as RC4, to produce

TABLE II: Summary of Notations

Notation	Definition
SK	Secret Key
N_o	Nonce
DK	Dynamic Key that has 512 bits length
m	The number of Vandermode matrices required by the proposed MIDA
K_M	Sub-key used to generate m IDA matrices
K_{SM}	Selection matrices sub-key used to construct the selection table SM used to select which of the IDA matrices will be used for each input sub-matrix
K_{USM}	Update selection table sub-key, which is used to construct the update permutation table π_{USM}
π_{USM}	Permutation table used to update the selection table SM for each new input log data
K_{SN}	Selection sub-key used to construct the selection table SN (which fragment for which node)
K_{USN}	Update selection sub-key used to construct the update selection table USN
K_{IA}	Sub-key used to ensure the message Integrity-Authentication of the produced fragments and it has 128 bits length
K_P	Sub-key used to construct permutation table π that it is used to encrypt of the collected log data
K_{UP}	Sub-key used to construct the update permutation table $u\pi$ that it is used to update the permutation table
π	Permutation table used for the proposed encryption scheme
$\|M\|$	Size of the original collected aggregated compressed log data
M	Original aggregated compressed log data M
Fragment	A final data fragment, which represents all columns of all the data shares (one row) . It is stored in one location storage entity
t	Number of fragments required for log recovery (threshold)
nsm	Number of sub-matrices inside initial data (M)
$Temp$	It is an initial vector of m values that repeated for $\lceil \frac{\|M\|}{m} \rceil$, where $Temp[i] = mod(i,\ m)$
X	The produced filtered keystream used to produced m IDA matrices
x_i	The i^{th} block of the produced filtered keystream X (has n bytes length) and it is used to construct the $G(i)$ IDA Vandermode matrix
G	A set of m dynamic IDA matrices
$G(i)$	The i^{th} dynamic IDA matrix
$G_t(i)$ and $G_t^{-1}(i)$	Any $t \times t$ of the i^{th} dynamic IDA matrix ($G(i)$) and its corresponding inverse matrix
$n, n \geq t$	Total number of fragments
MAC_i	It represents the MAC of the i^{th} fragment

a keystream. The iterations continue until each block of n bytes of the keystream has no zeros nor repeated values. This process leads to m blocks, $X_i = x_{i,1},\ x_{i,2},\ \ldots,\ x_{i,n}$ and $i = 1,\ 2,\ \ldots,\ m$. Each block is used to construct an IDA matrix.

Then, each element j of a block X_i $(x_{i,j})$ is used to construct a row of the Vandermonde matrix: $[1\ X_{i,j}\ x_{i,j}^2\ \ldots\ x_{i,j}^t])$, where $j = 1,\ 2,\ \ldots,\ n$. As such, we obtain $(n \times t)$ IDA matrix of the Vandermonde matrix form. Similarly, m IDA matrices are constructed for the IDA encoding process, and they are stored in memory. According to the Vandermode matrix properties, any t rows, of any IDA Vandermonde matrix, form an invertible $(t \times t)$ matrix G_t, which is a necessary condition to build the inverse of the modified IDA matrices.

Moreover, to avoid the regeneration of this set of m matrices at each new input log, the SM table is updated using π_{USM}. This results in selecting different m IDA matrices in a pseudo-random manner. If the fog nodes are limited in terms of computation and resources, the set of produced IDA matrices G can be updated, by permuting the row of each IDA matrix $G(i), i = 1,\ 2,\ \ldots,\ m$, using the selection table SN, which has values between 1 and n.

3) Dynamic Selection and Update of IDA matrices: The K_{SM} sub-key (see Fig. 3) is used to produce the selection table of the IDA matrices, SM. The construction of this selection table is based on a permutation table π_{SM}, which is generated by using the k_{SM} sub-key. In this specific case, $\pi_{SM}(i)^{th}$ serves to select the suitable i^{th} IDA matrix for the i^{th} encrypted data block. To produce the selection table, we first construct a primary table, $Temp$, with nsm elements. Each table element is equal to its index modulo m ($Temp[j] = mod(j, m)+1$ with $j = 1,\ 2,\ ldots,\ nsm$). Now, $Temp$ has nsm elements and their values vary between 1 and m, where m represents the number of produced IDA matrices and $m \leq nsm$. $Temp$ is permuted by using π_{SM} (that consists also of nsm elements) to produce the SM selection table and $1 \leq SM(i) \leq m$. At the decryption phase, SM serves also to select the inverse IDA matrix $G_t{-}1(\pi_{SM}(i))$, where G_t is a square matrix $(t \times t)$ and represents a set of unique t rows of G. In fact, SM controls the modified and its corresponding inverse MIDA processes.

Moreover, SM is updated after each new input of log contents by using another permutation table π_{USM}. π_{USM} is generated through the use of k_{USM} sub-key.

4) Fragments Distribution Tables: Additional permutation tables are needed for the distribution of the data fragments over the n corresponding entities. SN is generated by using a k_{SN} sub-key, and it consists of n elements. It also controls the distribution of these encrypted fragments over the n entities.

Moreover, an update table for the fragments distribution table USN is generated using k_{USN}. It is used in a similar manner to π_{USM} to update SN. As a result, the order, in which the diffused fragments are distributed over the $(n-1)$ neighbours, is changed for every application session.

VI. Securing Logs

In this section, we detail the steps of our proposed method to preserve the security of IoT logs. These steps include: normalization, aggregation, compression, encryption and MIDA encoding.

A. Aggregation and Normalization

In this step, the logs are first normalized and then aggregated. For normalization, the logs are converted into common syntax logs, an example of which is shown in Fig. 4. In the aggregation process, we group together the normalized logs of the same type, as shown in the example of Fig. 5.

B. Compression

The proposed solution aggregates the collected logs and compresses them using a lossless compression technique as indicated in the background part. The main goal is to reduce the input length, which in turn reduces the required computation, resources, communication and storage overhead. In this step, the Zstandard lossless compression algorithm is used. It combines a dictionary-matching stage (LZ77) with a large search window, in addition to a fast entropy coding stage. It is designed to offer compression ratios better than those offered by the Zip and gzip lossless algorithms, which are faster techniques.

Moreover, compressed data have better randomness and uniformity levels compared to the original uncompressed data. This helps in the design of a lightweight cipher scheme that requires only one round and one operation. Permuting and compressing logs remove any useful information about the original data in the logs.

C. Encryption Process

The encryption process is based on a dynamic permutation operation. First, the contents of an input log, M, are reshaped into a row form Ml of l bytes, where $Ml = \{m_1,\ m_2,\ \ldots,\ m_l\}$. Then, The permutation is applied to Ml using a dynamic permutation table π. The proposed cipher scheme exhibits linear computational complexity $O(l)$ and consequently, low execution time and resources overhead [40], [41]. The employed permutation table π of dimension l can be defined as: π=$[p_i]_{1\leq i\leq l}$.
A plain-text Ml of length l is given by: $Ml = [Ml_i]_{1\leq i\leq l}$.
After permutation, $C = Permutation(Ml,\ \pi)) = [Ml_{\pi_i}]_{1\leq i\leq \alpha}$.

Note that, for each session, the dynamic permutation table π is updated based on another permutation table, $U\pi$. Thus, each new log input is permuted differently (new permutation table π) compared to the previous or next logs.

D. The Modified IDA Process

As shown in Algorithm 1, the proposed MIDA algorithm takes as input, an aggregated, compressed, permuted and encrypted vector of data, D. This data vector is then fragmented into t fragments $F_1,\ \ldots,\ F_t$, each with a length of $nr = \lceil \frac{|M|}{t} \rceil$ elements. Then, these fragments are encoded into n diffused fragments $DF_1,\ \ldots,\ DF_n$. The IDA encoding process is based on a multiplication operation between a set of t fragments (matrix form) and a chosen IDA matrix $(n \times t)$, as illustrated in Fig. 6.

The proposed MIDA process regroups these t fragments into nsm sub-matrices, where each sub-matrix has $(t \times l)$ elements and F_i is the ith sub-matrix. The value of $nsm = \lceil \frac{|M|}{t \times l} \rceil$, and padding is applied when needed.

For each $SF_i,\ i = 1,\ 2,\ \ldots,\ nsm$ is multiplied by one of the m produced IDA matrices. As per its definition, the selection table SM has nsm elements, where each element has a value varying between 1 and m. In fact, $SM(i)$ is an integer value between 1 and m and $1 \leq i \leq nsm$. $SM(i)$ is used to select the dynamic encoding IDA matrix (m different IDA matrices), given the fact that they have a size of $(n \times t)$. Additionally, an IDA matrix can be used to encode more than one sub-matrix. In the presented algorithm, $G = GS(SM(i))$ means that $SM(i)$ IDA matrix is used. Next, the obtained sub-matrices are grouped into a matrix of $(n \times nr)$. Each row represents a diffused fragment, and hence, the output of the MIDA process includes n diffused fragments.

The values of t and n are related to the availability and security level of the proposed MIDA. Increasing n leads to additional communication and storage overhead. In this paper, t is chosen to be 4. Also, t and n can be changed according to the number of neighbouring nodes (fogs or IoT aggregation nodes).

Algorithm 1 Proposed MIDA (sub-matrix level).

1: $F = SF_1\ , SF_2, ,\ \ldots\ , SF_{nsm}$
2: **for** $i = 1 \rightarrow nsm$ **do**
3: $\quad G_i = G(SM(i))$
4: $\quad DF_j = G_i \odot SF_1)$
5: **end for**
6: $DF \leftarrow reshape(DF_1||DF_2||\ldots||DF_{nsm},\ n,\ nr)$

In fact, the proposed scheme preserves the homomorphic properties of the obtained fragmented data (addition and multiplication by scalar), since linear operations were used during ciphering or MIDA. This allows the processing of data without any privacy breach.

Raw Events				
Oct 29 10:10:10 type=login username=Petter IP address=10.10.10.10 destination= Gmail Message=Authenticate OK				
Oct 29 @10:10:20 User Jack logged in successfully at Gmail from IP 10.10.10.20				
Normalized Events				
ID	Source	Destination	Type	Time
1	10.10.10.10	1.1.1.1	Login	10:10:10
2	10.10.10.20	1.1.1.1	Login	10:10:20

Fig. 4: Normalization Example

Raw Events						
ID	Source	Destination	Type	Start Time	End Time	Count
10	10.10.10.10	20.20.20.20	ACL Request	10:10:00	10:10:10	1
10	10.10.10.10	20.20.20.20	ACL Request	10:10:10	10:10:20	1
10	10.10.10.10	20.20.20.20	ACL Request	10:10:20	10:10:30	1
10	10.10.10.10	20.20.20.20	ACL Request	10:10:30	10:10:40	1
Aggregated Events						
ID	Source	Destination	Type	First Time	Last Time	Count
10	10.10.10.10	20.20.20.20	ACL Request	10:10:00	10:10:40	4

Fig. 5: Example of raw events aggregation

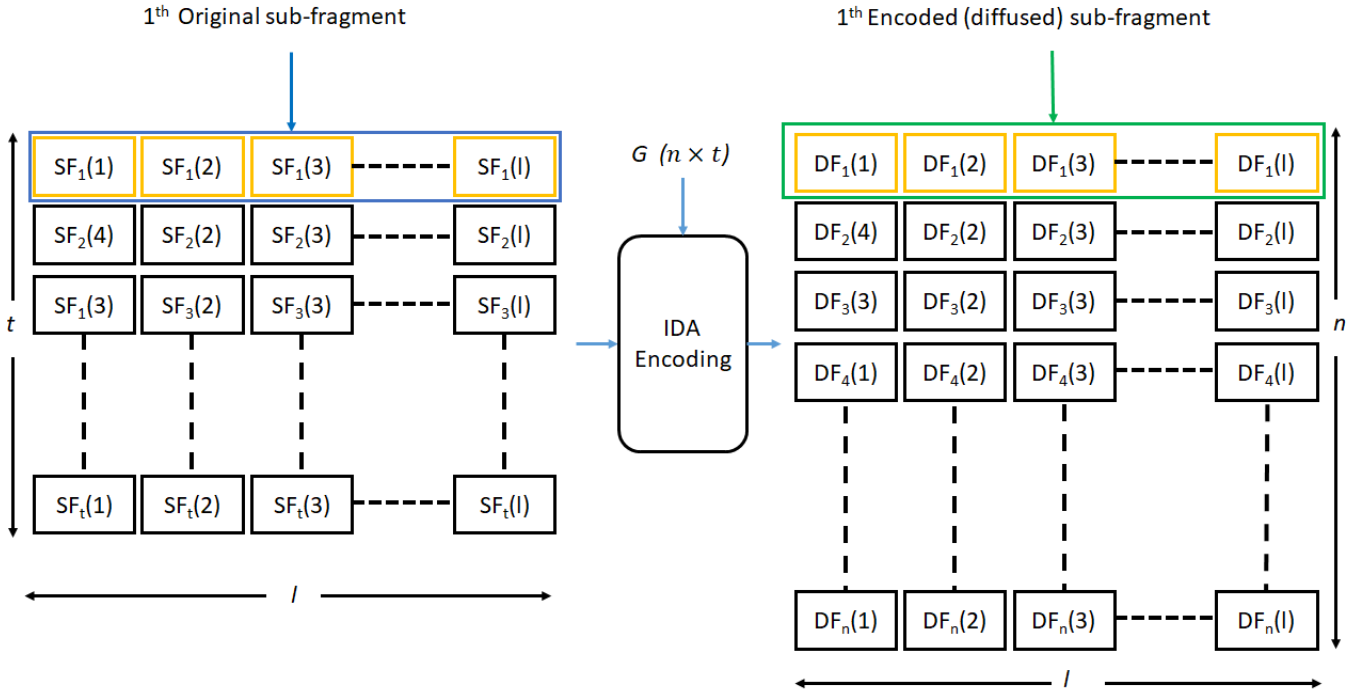


Fig. 6: Example for MIDA at the sub-matrix level for t=3 and n=5: Encoding of the t original sub-fragments (left) SF_j and $j = 1, 2, \ldots, t$ that transformed into n diffused sub-fragments DF_i (right) after IDA processing and $i = 1, 2, \ldots, n$.

E. Logs Authentication Process

In the proposed scheme, the authentication process of log contents is put in place for source authentication and data integrity. This can be done by using a keyed hash function such as HMAC [42] or by employing authentication operation mode such as CMAC [43], [44]. After obtaining the diffused fragments (output of the encoding of the initial data), $HMAC$ is applied on every single fragment through the use of the K_{IA} sub-key. As a result, for each fragment, f_i, a signature MAC_i is generated, and concatenated to the fragment, before dispersing the fragments over the n fog nodes or IoT aggregation nodes.

F. Inverse Cryptographic Solution

In case of an incident requiring the recovery of the log contents, for a specific time interval, a request is made to collect t diffused fragments of the corresponding IoT devices logs from the corresponding aggregation or fog nodes. Then, the inverse cryptographic process is applied. First, all cryptographic primitives are reproduced by using the corresponding $Nonce$ and DK, and their corresponding inverses are computed.

1) Any t diffused fragments are collected from t different entities. Then, the verification of the fragments is checked via the authentication process. The data integrity and source authentication of each of the fragments is validated by computing its MAC and comparing it with the sent one. This MAC value is computed via the same authentication algorithm that was applied during the protection phase.
2) The verified fragments are concatenated in a matrix, where each row represents $(nsm \times l)$ bytes. Then, this matrix is divided into nsm sub-matrices, each with a dimension of $(t \times l)$. Next, each sub-matrix is multiplied by its corresponding inverse IDA matrix, which is selected based on the SM table. After de-fragmenting all the encoded sub-matrices, we get a matrix of $t \times (nsm \times l)$, whereby the rows are reshaped to form a vector of the permuted log contents. Then, the inverse permutation process is applied using the inverse permutation table π^{-1}. Next, the original compressed log contents are recovered. Finally, the decompression algorithm is applied to recover the originally aggregated log contents.

VII. Security Analysis

An efficient cryptographic solution should protect data against the most known types of confidentiality, integrity and availability attacks

Typically, the diffused fragments must exhibit a high level of randomness and uniformity to prevent attackers from disclosing any useful information, or recognizing any pattern, from the transmitted logs, by using either statistical, differential, chosen/known plain-text, or brute-force attacks [39]. To that end, we perform different tests to quantify the strength of the proposed solution against data and key-related attacks [45], [46]. In these tests, the proposed scheme is considered as a black box. A set of initial messages are randomly chosen, each having a size of 1024 bytes. The tests include recurrence and correlation coefficient tests for evaluating the randomness and independence properties. Moreover, the probability density distributions of the obtained fragments are analyzed to assess the uniformity property. Then, the key sensitivity and the difference between the original and fragmented encrypted diffused fragments are computed. These tests and the corresponding results are detailed below.

A. Randomness

The encrypted fragments should achieve a high level of randomness to preserve the confidentiality and privacy of the logs' contents. This can done by applying the recurrence test and the correlation test between the original and the encrypted fragments (i.e independence test). Note that these tests are repeated $1,000$ times, with $n = 8$ and $t = 4$.

1) Recurrence: The recurrence test measures the temporal variation of a sequence $x = \{x_1,\ x_2,\ \ldots\}$ at two instants of time (t and $t + \Delta t$) [46]. This test consists of plotting the values of a sequence x at instant $t + \Delta t$ in function of the values of this sequence at instant t ($x(t + \Delta t) = f(x(t))$. In Fig. 8, the recurrence variation of the original and encrypted fragments (first fragment) is shown. The results show that the produced encrypted fragments exhibit a random recurrence.

2) Independence: The encrypted and original fragments should be independent [39]. This can be measured by performing several statistical tests such as the correlation coefficient and the percentage of the difference between the original and encrypted fragments, at the bit level.

First, the correlation coefficients are calculated between the original fragments and their corresponding encrypted ones. Then, the correlation coefficients among the different encrypted fragments are computed.

Fig. 9-a) shows the results of the Probability Density Function (PDF) of the obtained correlation coefficient matrix between the obtained fragments and the original ones. The results indicate that the correlation values are very close to the desired value of (0), and the distribution has a mean equals to 2.5889e-05 and a low standard deviation equals to 0.008. This confirms the independence of the encrypted fragments from the original log contents.

Second, the percentage of the difference, at the bit level, between the original fragments (a set of t encoded fragments) and their n encrypted fragments are computed. The results, presented in Fig. 9-b), show that the values of the percentage differences are very close to the ideal value of (50%), and the distribution has a mean equals to 49.99% and a low standard deviation of 0.13.

Using a random dynamic key, we illustrate with numerical examples, the inter-correlation and difference percentage at the bit level between the original and obtained fragments. The results are shown in TABLE III and TABLE IV, respectively. Similar results among the obtained fragments are shown in TABLE V and TABLE VI.

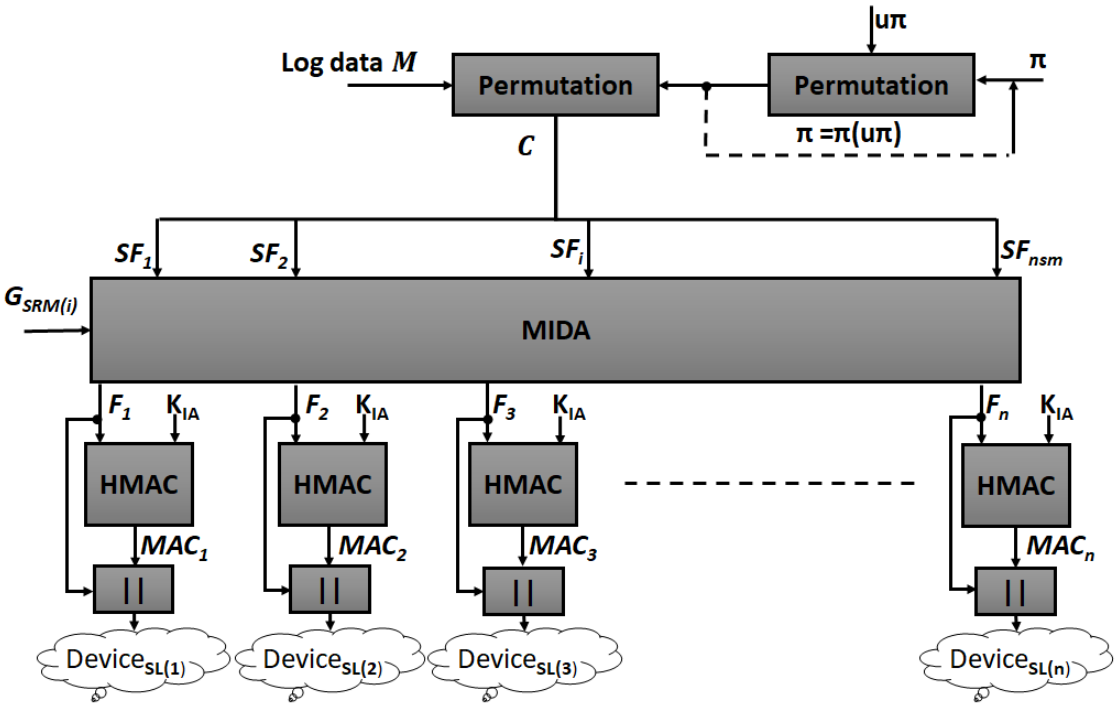
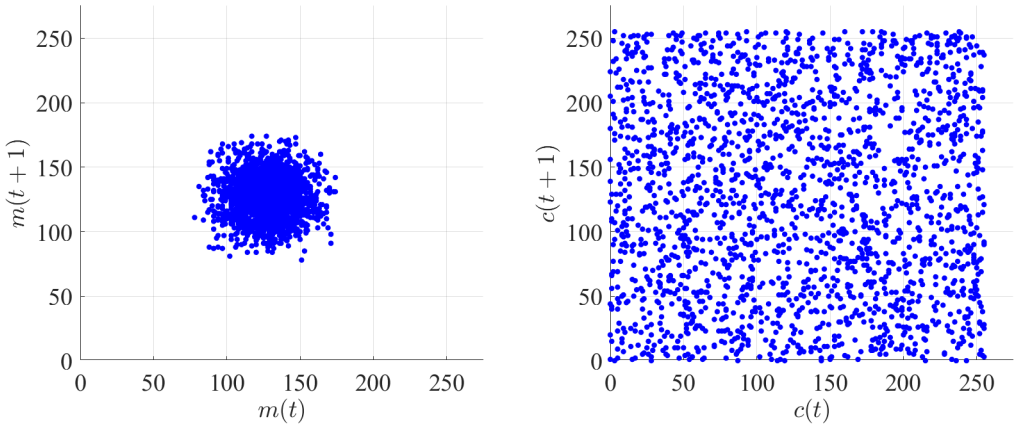


Fig. 7: Proposed Log Protection Cryptographic Solution



(a) (b)

Fig. 8: Recurrence of original and its corresponding encrypted fragmented one (2048 bytes length), (t=4 and n=8).

TABLE III: Correlation coefficient between the original and the encoded/encrypted fragments for a random dynamic key with t =4 and n =8

	EF_1	EF_2	EF_3	EF_4	EF_5	EF_6	EF_7	EF_8
OF_1	-0.0018	-0.0014	0.0055	0.0005	-0.0244	-0.0059	0.0026	0.0004
OF_2	-0.0149	0.0024	-0.0178	-0.0130	0.0139	0.0023	0.0204	0.0115
OF_3	-0.0026	-0.0239	0.0261	-0.0014	0.0039	-0.0275	0.0344	0.0121
OF_4	0.0062	0.0083	0.0150	-0.0190	-0.0067	0.0004	-0.0041	0.0202

B. Uniformity

To thwart statistical attacks, the uniformity property needs to be met by the proposed scheme. In fact, the uniformity property can be visually quantified and verified when the distribution of the encrypted fragments is close to a uniform distribution. In Fig. 10, the distribution of the original

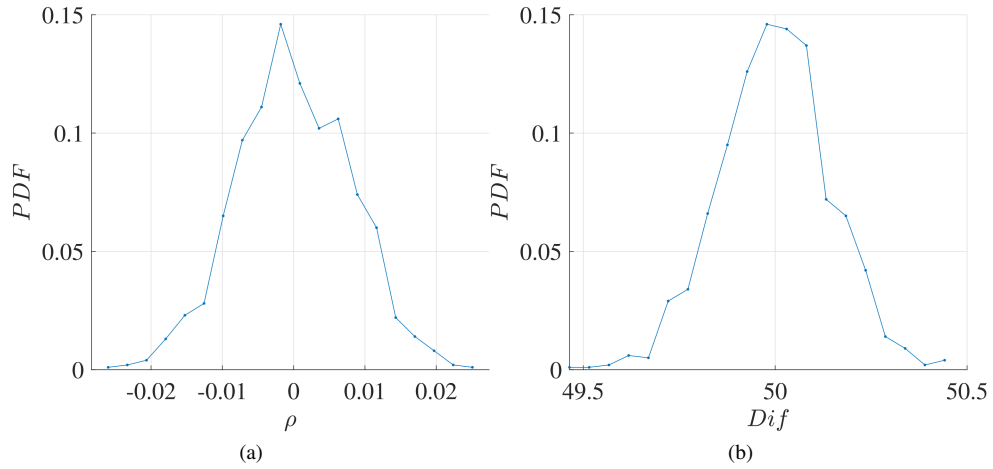


Fig. 9: Correlation coefficients between original and fragmented data(a). In addition, the difference between original and encrypted fragmented data (% of changed bits) (b) for k=4 and n=8.

TABLE IV: The percentage difference between the original and the encoded/encrypted fragments for a random dynamic key with t =4 and n =8. Average equals to 49.97% and standard deviation is equal to 0.1219.

	EF_1	EF_2	EF_3	EF_4	EF_5	EF_6	EF_7	EF_8
OF_1	49.9344	50.3125	50.1812	49.7031	49.7781	50.1750	49.8250	49.7312
OF_2	49.9188	50.2031	49.6781	49.7125	49.9125	50.2406	50.1844	50.1219
OF_3	50.5563	49.8094	49.8844	49.7625	49.5187	49.9906	49.6656	50.0906
OF_4	49.4844	50.4063	49.4875	50.3094	50.6906	49.9813	50.2188	49.8312

TABLE V: Correlation coefficient among encrypted fragmentation

	EF_1	EF_2	EF_3	EF_4	EF_5	EF_6	EF_7	EF_8
EF_1	-							
EF_2	-0.0097	-						
EF_3	0.0059	0.0143	-					
EF_4	0.0263	-0.0041	0.0196	-				
EF_5	-0.0124	-0.0281	0.0257	-0.0272	-			
EF_6	0.0044	-0.0305	0.0186	-0.0010	-0.0267	-		
EF_7	0.0094	-0.0246	-0.0121	0.0383	0.0019	-0.0058	-	
EF_8	-0.0401	0.0225	0.0331	0.0126	-0.0130	0.0225	-0.0085	-

TABLE VI: The percentage bit difference among encrypted fragmentation with t =4 and n =8

	EF_1	EF_2	EF_3	EF_4	EF_5	EF_6	EF_7	EF_8
EF_1	-							
EF_2	50.2156	-						
EF_3	49.9656	49.8125	-					
EF_4	50.3563	49.9781	49.7594	-				
EF_5	50.0062	50.4094	50.0594	50.4063	-			
EF_6	50.0031	49.9125	49.7687	49.8781	50.1219	-		
EF_7	49.8656	50.0375	49.8813	49.3469	49.9406	49.9437	-	
EF_8	50.2719	49.5625	49.9563	49.6969	50.4594	50.0125	50.1187	-

fragments, along with their corresponding encrypted ones is presented. The distributions show that the contents of the encrypted fragments are uniformly spread over the entire space.

C. Sensitivity

The proposed cryptographic scheme should ensure a high level of key sensitivity to guard against key-related attacks. In the proposed scheme, all cryptographic primitives depend on the generated dynamic key, DK, and as such, a small change in the secret key would result in completely different set of encrypted fragments.

To evaluate the dynamic key sensitivity, two dynamic secret keys, DK_w and DK'_w, differing by only one random bit, are used to encrypt the same log contents, with $w = 1, \ldots, 1,000$. The sensitivity measure for the w^{th} dynamic

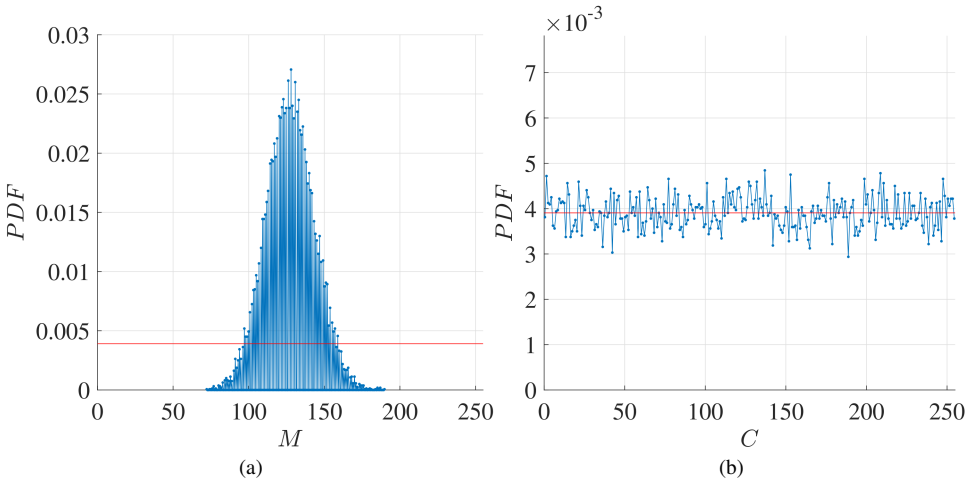


Fig. 10: The distribution of an original fragment (a) and its correspondent encrypting encoded one (t=4 and $n = 8$)

key (DK'_w) is calculated as follows:

$$KS_w = \frac{\sum_{it=1}^{T} S_{DK_w} \oplus S_{DK'_w}}{l} \times 100\% \quad (6)$$

Where S represents the proposed MIDA cryptographic solution and T is the length of the input data in bits. We run the sensitivity test for 1,000 iterations. At each iteration, different messages and different keys are used, with the condition of having a single bit changed between the two considered keys. As shown in Fig. 11, the PDF of the obtained key sensitivity percentages indicates that the obtained values are very close to the mean value of 50.58% and the standard deviation is very low equals to 0.141. These results show that the proposed scheme exhibits a high dynamic key sensitivity. In addition, for a random key, the inter-difference percentages among encoded fragments is shown in TABLE VII. This shows that a high difference is also reached between any pair of encoded fragments. Moreover, the correlation test is applied on each pair of encoded fragments and the obtained results confirm the independence among encoded fragments, as shown in TABLE VIII.

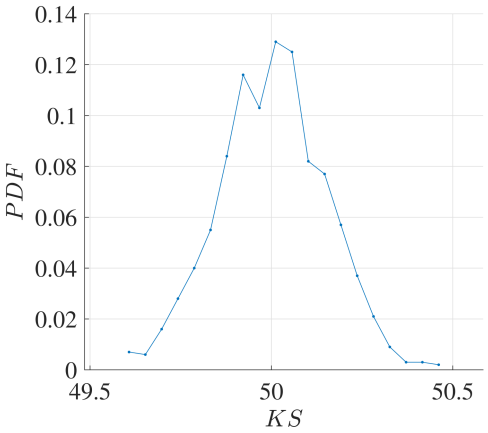


Fig. 11: Average Key sensitivity test measuring the bit difference between two sets of encrypted diffused fragmentation obtained for the same data but with one slightly different in the dynamic key for 1000 iterations

VIII. Cryptanalysis Discussion

In the following subsections, we discuss the immunity of the proposed scheme against different attacks, including statistical, differential, chosen/known plain-test and brute-force attacks. To do so, we consider the situation where k encrypted diffused fragments of log contents were revealed to an eavesdropping attacker. In addition, we assume that the proposed fragmentation scheme is known to the attacker.

A. Statistical Attacks:

This attack targets the encrypted fragments to reveal some statistical properties. To prevent this kind of attacks, the diffused encrypted data should satisfy the randomness and uniformity properties, which was confirmed by the results in Section VII, including the entropy analysis, probability density function and correlation tests.

B. Brute-Force attack:

To prevent brute-force attacks, the size of the secret and dynamic keys should be at least 128 bits. In the proposed scheme, the master secret key size can be 128, 196 or 256 bits, whilst the dynamic key size is 512 bits.

C. Known & Chosen Plain/Cipher-Text Attacks:

A different set of cryptographic primitives are used at each session for the collected log contents. Therefore, different sets of encrypted diffused fragments will be obtained for the same input. Consequently, this makes any known or chosen plain/cipher text attack unfeasible.

TABLE VII: Key sensitivity test between two obtained fragmented sets obtained for the same data but with two slightly different keys (Dk and DK') with t =4 and n =8

	ED_1	ED_2	ED_3	ED_4	ED_5	ED_6	ED_7	ED_8
ED'_1	50.08	49.64	49.89	50.25	50.07	50.14	50.14	50.68
ED'_2	49.98	49.94	49.83	49.82	50.40	50.05	50.08	50.23
ED'_3	50.47	49.93	49.63	50.42	49.77	50.41	49.82	50.07
ED'_4	49.69	50.08	49.61	50.32	49.86	50.53	50.03	50.11
ED'_5	50.52	50.34	50.50	50.04	50.53	49.67	49.72	50.0
ED'_6	49.60	50.23	49.68	50.52	49.77	49.73	50.05	50.10
ED'_7	49.81	50.23	50.27	50.13	50.08	50.58	49.59	50.0
ED'_8	50.14	50.14	50.06	50.19	49.84	50.17	49.68	50.29

TABLE VIII: Correlation coefficient between two obtained fragmented sets obtained for the same data but with two slightly different keys (Dk and DK') with t =4 and n =8

	ED_1	ED_2	ED_3	ED_4	ED_5	ED_6	ED_7	ED_8
ED_1	0.0088	-0.0210	-0.0157	-0.0013	-0.0094	-0.0014	-0.0245	-0.0126
ED_2	0.0161	0.0242	-0.0085	0.0283	-0.0085	0.0079	-0.0079	0.0173
ED_3	-0.0043	0.0021	0.0193	0.0022	-0.0100	-0.0037	0.0033	0.0166
ED_4	-0.0131	0.0135	-0.0135	-0.0017	-0.0147	0.0003	0.0037	0.0100
ED_5	-0.0195	0.0035	0.0036	0.0063	0.0142	-0.0195	-0.0036	0.0289
ED_6	-0.0113	-0.0122	0.0041	-0.0150	0.0132	-0.0006	-0.0221	-0.0158
ED_7	0.0094	-0.0072	0.0228	0.0296	-0.0428	0.0187	0.0267	-0.0014
ED_8	-0.0231	-0.0107	0.0127	-0.0170	0.0144	-0.0231	-0.0080	-0.0161

D. Linear and Differential attacks

The dynamic cryptographic primitives are updated for each new input of log contents. This eliminates any similarity among the resulted encrypted encoded fragments. According to Section VII-C, a high key sensitivity was achieved by the proposed solution. Therefore, any single bit change in the secret key or in the Nonce leads to different cryptographic and update primitives. Moreover, the existing cryptanalysis techniques that target static cryptographic algorithms cannot break the proposed solution that update its corresponding cryptographic primitives after each new input log data.

IX. Performance Analysis

In order to protect the log files, the proposed solution introduces a cost overhead in terms of computations (and consequently delay and resources) in addition to communication and storage costs. All of these performance measures are discussed and assessed in this section. The primary objective of the proposed solution is to strike a good balance between the system performance and security level.

A. Computational Overhead

In this section, we compute the overhead costs of the proposed dynamic key derivation function and the generation of cryptographic primitives, in addition to the update process of the cryptographic primitives.

The key derivation function requires only two operations to produce the dynamic key for each new session:

1) XOR operation between the nonce and the secret key.
2) One iteration of a secure hash function for one input block (512 bits, if SHA-512 is used).

Then, a set of dynamic sub-keys is obtained from the dynamic key, and these are used to generate all the required cryptographic and update primitives.

The computational complexity of the dynamic key generation scheme CD_{DKG}, cryptographic and update primitives are described below.

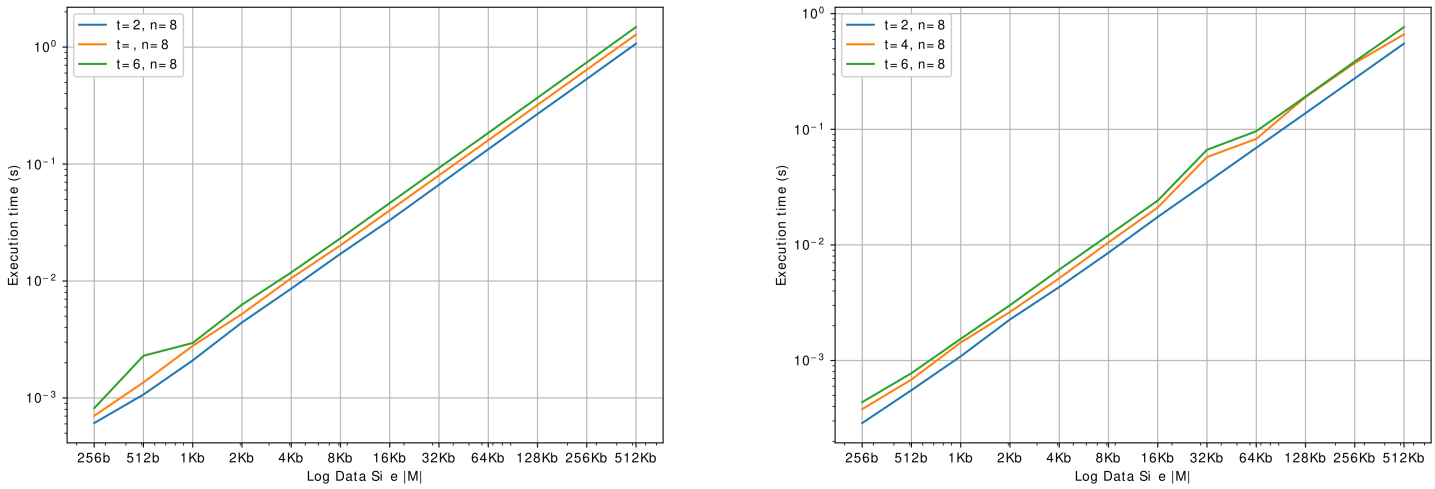
1) T_{xor} denotes the required "exclusive or" logical operation time for two blocks of the same length.
2) T_H denotes the required hash time for a block of h bytes.
3) $T_{MKSA}(l)$ denotes the required execution time of the modified KSA of RC4 for a table with l elements. It exhibits a low computational delay and it is used to construct the permutation tables.
4) T_{CIDA} denotes the required time to construct m IDA matrices, each with a size of $n \times t$.

$$CD_{DKG} = T_{xor} + T_H + 2 \times T_{MKSA}(n) + 2 \times T_{MKSA}(nsm) \\ 2 \times T_{MKSA}(|M|) + T_{CIDA} \tag{7}$$

In order to reduce the overhead associated with the dynamic key derivation function and the construction of the cipher primitives, the selection and permutation tables are updated in a lightweight manner by simply using permutation operations. The computational complexity of the update cryptographic primitives is described by:

$$CD_{UPP} = T_\pi(|M|) + T_\pi(|n|) + T_\pi(\frac{|M|}{t})$$

where $T_\pi(x)$ denotes the required permutation time for a table of length x. Note that the permutation process exhibits linear computational complexity $O(x)$, which makes the overhead of the update process lower than that of regenerating the cryptographic primitives (Eq.7).



(a) RPI0

(b) RPI3

Fig. 12: Variation of the average execution time of cryptographic solution versus log data length ($|M|$) for n=8 and for different values of t for RPI0 (a) and RPI3 (b).

The encryption scheme requires a single iteration including one permutation operation. Moreover, a look-up table can be used to replace the multiplication operation in the IDA process, which reduces the computational overhead. Accordingly, the proposed solution is lightweight and it can be adapted according to the entity's (fog or cluster IoT node) limitations in terms of power, storage and computations.

Moreover, the delay overhead of the proposed solution can be optimized if the fog entity can perform parallel computations. Mainly, the encryption/decryption process can be realized in parallel and separately at the fragment level. In this case, t permutation tables ($\frac{M}{t}$ elements) can be used, where each one is used to permute a fragment. Moreover, the proposed MIDA process can also be performed in parallel at the sub-matrix level; each sub-matrix can be encoded and decoded independently from the others. Similarly, the n diffused encrypted fragments can also be authenticated in parallel.

Therefore, the computational complexity $CD_{proposed}$ of the proposed cryptographic scheme is described in the following:

$$CD_{proposed} = T_{\pi}(|M|) + n \times T_{Auth} + nsm \times T_{IDA}(t,n) \quad (8)$$

where:

1) T_{Auth} denotes the required execution time of the HMAC algorithm, for a fragment of $\frac{|M|}{t}$ bytes.
2) T_{IDA} denotes the required IDA diffusion time for a set of t input fragments that produce n output diffused fragments.

Note that the encryption and message authentication processes can be performed in parallel at the fragment level. The length of each fragment is $\frac{|M|}{t}$ bytes. Consequently, the proposed solution is efficient based on the overall reduced overhead.

B. *Execution Time*

The efficiency of the proposed modified IDA scheme is also analyzed in terms of execution time. The solution is implemented with an optimized Galois Field Arithmetic library scheme. The proposed cryptographic solution was executed on different Raspberry Pi devices (0 and 3) that can be considered as fog nodes (called RPI0, and RPI3, respectively). RPI0 has 512MB RAM and 1GHz mono-core micro-controller (ARMv6 instructions), while RPI3 has a 1.2GHz quad-core microprocessor (ARMv7 instructions) and 1GB RAM. RPI3 is more powerful compared to RPI0.

In Fig. 12, the average required execution time of the proposed cryptographic solution is presented for RPI0 and RPI3 in function of input log data size $|M|$, in addition to n and t. The results show that increasing the size of the input leads to an increase in the execution time. Increasing n will also increase the execution time for a fixed size of input data log. Similarly, increasing t will lead to an increase in the execution time for a fixed size of input data and n.

Compared to other secret sharing variants, as presented in TABLE IX, the proposed solution exhibits a linear computational complexity; the execution time variation is linear as a function of the message length, in contrast to the polynomial variation as in the original Shamir secret sharing variant. This makes the solution suitable for constrained IoT or fog devices. In addition, the proposed MIDA outperforms the original IDA,

TABLE IX: Comparison among several secret sharing variants

	Communication Overhead	Key Management	Operation Base	Revoking Authority Optimized	Data Deduplication	Security Services	Computational Overhead
Secret Sharing Shamir	$(n-1) \times \|M\|$	Keyless	Polynomial	No	No	DA	None
IDA	$(n-t) \times \frac{\|M\|}{t}$	Keyless	Matrix	No	Yes	DA and Weak DC	-
AONT-RS	$((n-t) \times \frac{\|M\|}{t} + \frac{\|H(C)\|}{t})$	Xored with data	Polynomial	No	No	DA, DC, DI and SA	Systematic IDA + Encryption + PRNG(1) + Hash
IDA + Over Encryption	$((n-t) \times \frac{\|M\|}{t})$	NA	Polynomial	Yes	No	DA and DC	IDA + Partial Encryption

which cannot be parallelized at the sub-fragment level. Thus, if parallel computing is possible, increasing the number of threads will lead to further reduction of the required execution time. Let us indicate that the proposed cryptographic solution achieves a better performance on RPI3 since approximately double the input log data rate can be processed compared to RPI0.

C. Storage and Communication Overhead

The produced fragments' size is $n \times \frac{|M|}{t}$ bytes, which represents the storage overhead. In addition, the communication overhead is $(n+t) \times \frac{|M|}{t}$ bytes, given that the reconstruction phase exhibits a communication overhead of $t \times \frac{|M|}{t}$ bytes. In fact, the data redundancy, represented by the $(n-t) \times \frac{|M|}{t}$ bytes, is inevitable to prevent log contents loss in case of damage or alteration. However, when the value of n is close to t, the storage overhead and availability level decrease. In contrast, when t is less than n, data availability degree increases, associated with a higher communication overhead. The choice of n depends on the required availability level $(n-t)$. The values of n and t must be set to reach a good balance between data availability and communication overhead. The communication and storage overhead ensures logs' availability to thwart link failure or channel errors in case the IoT devices or fog nodes are compromised. In the proposed scheme, $(n-t)$ redundant encoded rows are produced and transmitted to overcome data loss, damage and alteration.

D. Efficiency

The proposed scheme utilizes a cipher with one round and one permutation operation, in addition to an optimized IDA variant to protect the IoT log files. The protection is performed at the fog node. This means that no additional operations are required at the destination storage nodes. In addition, the proposed message authentication step aims to validate the integrity and authentication of the received encrypted encoded fragment before decrypting it. Thus, the proposed solution provides multiple security services (data confidentiality, source authentication and data integrity in addition to data availability) with a low computational complexity and overhead.

X. Conclusion

Logs are key evidence in any forensics investigation process, especially, for IoT forensics where new security challenges emerge, making the security of IoT devices logs critical. In this paper, for the first time, a scheme is proposed to ensure the IoT logs availability, confidentiality, authentication and integrity. The proposed scheme relies on the dynamic key approach with lightweight cryptographic functions to ensure a high security level and low computational and storage requirements. Modifying the standard IDA scheme, the proposed MIDA aims at fragmenting logs into n fragments to be distributed over n IoT devices or fog nodes. This complicates the attacker's task in compromising n devices or communication channels. Several statistical tests were performed to confirm the robustness of the dynamic key-based approach towards defending against statistical attacks. In addition, the security analysis proved that the proposed method guards well against known attacks. Finally, the performance analysis shows that the proposed solution requires low computational and storage resources compared to previous works.

Acknowledgement

This paper is supported with funds from the Maroun Semaan Faculty of Engineering and Architecture at the American University of Beirut and by the EIPHI Graduate School (contract "ANR-17-EURE-0002").

References

[1] Golden G Richard III and Vassil Roussev. Next-generation digital forensics. *Communications of the ACM*, 49(2):76–80, 2006.

[2] Ibrar Yaqoob, Ibrahim Abaker Targio Hashem, Arif Ahmed, SM Ahsan Kazmi, and Choong Seon Hong. Internet of things forensics: Recent advances, taxonomy, requirements, and open challenges. *Future Generation Computer Systems*, 92:265–275, 2019.

[3] M. Chernyshev, S. Zeadally, Z. Baig, and A. Woodward. Internet of things forensics: The need, process models, and open issues. *IT Professional*, 20(3):40–49, May 2018.

[4] Bernie Lantz, Rob Hall, and Jason Couraud. Locking down log files: enhancing network security by protecting log files. *Issues in Information Systems*, 7(2):43–47, 2006.

[5] Mihir Bellare. Forward integrity for secure audit logs. Technical report, 1997.

[6] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security (TISSEC)*, 2(2):159–176, 1999.

[7] Brent Waters, Dirk Balfanz, Glenn Durfee, and Diana K. Smetters. Building an encrypted and searchable audit log. In *NDSS*, 2004.

[8] Jason E Holt. Logcrypt: forward security and public verification for secure audit logs. 2006.

[9] Di Ma and Gene Tsudik. A new approach to secure logging. *Trans. Storage*, 5(1):2:1–2:21, March 2009.

[10] Attila Altay Yavuz and Peng Ning. Baf: An efficient publicly verifiable secure audit logging scheme for distributed systems. In *2009 Annual Computer Security Applications Conference*, pages 219–228. IEEE, 2009.

[11] Attila A Yavuz, Peng Ning, and Michael K Reiter. Baf and fi-baf: Efficient and publicly verifiable cryptographic schemes for secure logging in resource-constrained systems. *ACM Transactions on Information and System Security (TISSEC)*, 15(2):9, 2012.

[12] Attila A Yavuz, Peng Ning, and Michael K Reiter. Efficient, compromise resilient and append-only cryptographic schemes for secure audit logging. In *International Conference on Financial Cryptography and Data Security*, pages 148–163. Springer, 2012.

[13] Gunnar Hartung. Attacks on secure logging schemes. In *International Conference on Financial Cryptography and Data Security*, pages 268–284. Springer, 2017.

[14] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[15] Scott A Crosby and Dan S Wallach. Efficient data structures for tamper-evident logging.

[16] Vishal Karande, Erick Bauman, Zhiqiang Lin, and Latifur Khan. Sgx-log: Securing system logs with sgx. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 19–30. ACM, 2017.

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

[18] Arunesh Sinha, Limin Jia, Paul England, and Jacob R Lorch. Continuous tamper-proof logging using tpm 2.0. In *International Conference on Trust and Trustworthy Computing*, pages 19–36. Springer, 2014.

[19] Hung Nguyen, Radoslav Ivanov, Linh TX Phan, Oleg Sokolsky, James Weimer, and Insup Lee. Logsafe: Secure and scalable data logger for iot devices. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 141–152. IEEE, 2018.

[20] Carlton Shepherd, Raja Naeem Akram, and Konstantinos Markantonakis. Emlog: Tamper-resistant system logging for constrained devices with tees. *CoRR*, abs/1712.03943, 2017.

[21] Shams Zawoad, Amit Kumar Dutta, and Ragib Hasan. Seclaas: Secure logging-as-a-service for cloud forensics. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 219–230, New York, NY, USA, 2013. ACM.

[22] I. Ray, K. Belyaev, M. Strizhov, D. Mulamba, and M. Rajaram. Secure logging as a service—delegating log management to the cloud. *IEEE Systems Journal*, 7(2):323–334, June 2013.

[23] Alessandro Bellini, Emanuele Bellini, Monica Gherardelli, and Franco Pirri. Enhancing iot data dependability through a blockchain mirror model. *Future Internet*, 11(5):117, 2019.

[24] Adam Oliner, Archana Ganapathi, and Wei Xu. Advances and challenges in log analysis. *Communications of the ACM*, 55(2):55–61, 2012.

[25] Vidyasagar M Potdar, Muhammad A Khan, Elizabeth Chang, Mihaela Ulieru, and Paul R Worthington. e-forensics steganography system for secret information retrieval. *Advanced Engineering Informatics*, 19(3):235–241, 2005.

[26] Ken Chiang and Levi Lloyd. A case study of the rustock rootkit and spam bot. *HotBots*, 7:10–10, 2007.

[27] Russell Smith. *Least Privilege Security for Windows 7, Vista and XP*. Packt Publishing Ltd, 2010.

[28] Luyi Xing, Xiaorui Pan, Rui Wang, Kan Yuan, and XiaoFeng Wang. Upgrading your android, elevating my malware: Privilege escalation through mobile os updating. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 393–408. IEEE, 2014.

[29] Ola Salman, Imad Elhajj, Ayman Kayssi, and Ali Chehab. Edge computing enabling the internet of things. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 603–608. IEEE, 2015.

[30] Ola Salman, Imad Elhajji, Ali Chehab, and Ayman Kayssi. Iot survey: An sdn and fog computing perspective. *Computer Networks*, 2018.

[31] H. Rahman, N. Ahmed, and I. Hussain. Comparison of data aggregation techniques in internet of things (iot). In *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pages 1296–1300, March 2016.

[32] Joseph Azar, Abdallah Makhoul, Mahmoud Barhamgi, and Raphaël Couturier. An energy efficient iot data compression approach for edge machine learning. *Future Generation Computer Systems*, 96:168 – 175, 2019.

[33] J Uthayakumar, T Vengattaraman, and P Dhavachelvan. A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. *Journal of King Saud University-Computer and Information Sciences*, 2018.

[34] Robert A. McLeod, Ricardo Diogo Righetto, Andy Stewart, and Henning Stahlberg. Mrcz – a file format for cryo-tem data with fast compression. *Journal of Structural Biology*, 201(3):252 – 257, 2018.

[35] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, April 1989.

[36] Rasmus W Lauritsen. Backups with computational secret sharing. *University of Aarhus*, 9, 2008.

[37] William Stallings. *Cryptography and network security: principles and practice*. Pearson Upper Saddle River, 2017.

[38] PUB FIPS. 180-1. secure hash standard. *National Institute of Standards and Technology*, 17:45, 1995.

[39] Hassan Noura, Ali Chehab, Lama Sleem, Mohamad Noura, Raphaël Couturier, and Mohammad M Mansour. One round cipher algorithm for multimedia iot devices. *Multimedia tools and applications*, 77(14):18383–18413, 2018.

[40] Hassan Noura and Damien Couroussé. Lightweight, dynamic, and flexible cipher scheme for wireless and mobile networks. In *International Conference on Ad Hoc Networks*, pages 225–236. Springer, 2015.

[41] Peng Zhang, Yixin Jiang, Chuang Lin, Yanfei Fan, and Xuemin Shen. P-coding: secure network coding against eavesdropping attacks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[42] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. Rfc 2104: Hmac: Keyed-hashing for message authentication. *Internet Engineering Task Force*, 252, 1997.

[43] Stefan Marksteiner, Víctor Juan Expósito Jiménez, Heribert Valiant, and Herwig Zeiner. An overview of wireless iot protocol security in the smart home domain. In *2017 Internet of Things Business Models, Users, and Networks*, pages 1–8. IEEE, 2017.

[44] William Stallings. Nist block cipher modes of operation for authentication and combined confidentiality and authentication. *Cryptologia*, 34(3):225–235, 2010.

[45] Hassan Noura, Ali Chehab, Mohamad Noura, Raphaël Couturier, and Mohammad M Mansour. Lightweight, dynamic and efficient image encryption scheme. *Multimedia Tools and Applications*, pages 1–35, 2018.

[46] Hassan Noura, Steven Martin, Khaldoun Al Agha, and Khaled Chahine. Erss-rlnc: Efficient and robust secure scheme for random linear network coding. *Computer Networks*, 75, Part A:99 – 112, 2014.