Partitioning-Based Data Sharing Approach for Data Integrity Verification in Distributed Fog Computing

Uma Maheswari Kaliyaperumal[1,*], Mary Saira Bhanu Somasundaram[1], Nickolas Savarimuthu[2]

[1]Department of Computer Science and Engineering, National Institute of Technology, Tiruchirappalli, India

[2]Department of Computer Applications, National Institute of Technology, Tiruchirappalli, India

Received 11 August 2022; received in revised form 12 November 2022; accepted 13 November 2022

DOI: https://doi.org/10.46604/ijeti.2023.10685

Abstract

With the increasing popularity of the internet of things (IoT), fog computing has emerged as a unique cutting-edge approach along with cloud computing. This study proposes an approach for data integrity verification in fog computing that does not require metadata stored on the user side and can handle big data efficiently. In the proposed work, fuzzy clustering is used to cluster IoT data; dynamic keys are used to encrypt the clusters; and dynamic permutation is used to distribute encrypted clusters among fog nodes. During the process of data retrieval, fuzzy clustering and message authentication code (MAC) are used to verify the data integrity. Fuzzy clustering and dynamic primitives make the proposed approach more secure. The security analysis indicates that the proposed approach is resilient to various security attacks. Moreover, the performance analysis shows that the computation time of the proposed work is 50 times better than the existing tag regeneration scheme.

Keywords: fog computing, data integrity, cloud computing, fuzzy clustering, dynamic key

1. Introduction

Fog computing is a widely used virtualization platform that provides various services, such as storage, networking, and computing-related services between data centers and users. It helps to replace classical cloud computing since it supports latency-sensitive, geographically distributed, and quality-of-service-based internet of things (IoT) applications. It was developed by Cisco to extend cloud computing to the edge of its network. It reduces latency by combining fog servers as the interface between users and cloud servers. It can process a large amount of data with portable applications and be deployed on heterogeneous hardware. These properties make fog computing well-suited for various time and location-sensitive applications.

With cloud computing, data is outsourced to remote servers, and the data owners have no control over their data. The cloud service provider (CSP) may fail to keep or purposefully erase infrequently viewed data files that belong to regular cloud customers for their profit. Furthermore, the CSP may conceal data corruption caused by server hacks or byzantine failures. Hence, verifying the integrity of data stored in the cloud servers is a key constraint in this situation. To address this concern, data owners should implement mechanisms to maintain data integrity, regularly verify data possession, and detect illegal changes.

The traditional data integrity verification technique consists of two phases:

(1) Setup phase: The user uses the key generation algorithm to set the public and private parameters and the signature generation to preprocess the data file and compute the metadata. Then, the user uploads the file to the remote cloud server and sends the metadata to a third-party authority (TPA) for auditing.

---

[*] Corresponding author. E-mail address: ku.cse09@gmail.com

(2) Audit phase: During the time of the verification, the TPA sends a challenge message to the cloud server. After executing the GenProof algorithm, the cloud server will generate a response message from the stored data file. The TPA verifies the response using the verification metadata via the VerifyProof algorithm [1].

In both cloud computing and fog computing, data is not under the control of the data owner. Therefore, verifying the integrity of data stored on remote cloud/fog servers is critical for the proper functioning of the entire system. Several methods have been proposed in recent years to verify the integrity of data stored in cloud servers. However, most of them are not suitable for fog computing. In traditional cloud computing, the data owner creates the data, and they must generate the metadata for integrity verification. In fog computing, a variety of IoT devices create the data, while these devices cannot generate the metadata and verify them. The user who stores the data retrieves it in cloud computing. Various end-users access the data generated by IoT devices in fog computing. As a consequence, creating novel verification approaches for fog data integrity is crucial [2]. Fog nodes have limited power, storage, and processing capabilities. Traditional cryptographic techniques are unsuitable for fog computing since they consume more energy and memory.

On the other hand, a poorly designed lightweight solution would compromise the fog nodes. An effective approach for securing data storage is required to ensure data integrity while maintaining low latency, energy, and memory consumption [3]. The proposed work guarantees the integrity of fog data using a two-level integrity verification mechanism: cluster-level and block-level. The cluster-level verification uses the dynamic key and message authentication code (MAC) to verify the integrity of each cluster, whereas the block-level verification uses fuzzy clustering that does not require metadata for verification. The uniqueness of the proposed work is that block-level verification is performed without metadata, and the data blocks are divided into clusters of different sizes to prevent attackers from predicting the size of the clusters. The usage of dynamic keys, varying number of clusters, different cluster sizes, dynamic permutation tables, and random fog storage node selection make the attacker's task more difficult. The proposed work offers higher efficiency in maintaining data integrity, and it is the best suitable approach for handling a large amount of data where most existing schemes fail.

The remainder of the paper is organized as follows: Section 2 covers the existing data integrity verification techniques. Section 3 presents the proposed partitioning-based data integrity verification approach, and section 4 discusses the security and performance analysis of the proposed approach. Section 5 concludes the study with future enhancements.

2. Related Work

When data is outsourced to remote servers, preserving its integrity is a major concern. In this concern, many cryptography-based schemes have been studied by researchers in recent years. Ateniese et al. [4] proposed a scheme based on RSA-homomorphic authenticators that allows an unlimited number of challenges and public auditing to prove data integrity in cloud storage. Researchers propose various authentication structures to support integrity verification on cloud data. Wang et al. [5] proposed a verification scheme based on the Merkle hash tree (MHT). In this scheme, the client generates a random signing key pair and the MHT root, where the leaves of the MHT are hashes of the file blocks. The client generates block signatures and sends them along with the signed root to the server. The client initiates data integrity verification by picking random blocks and sending them to the server. The client verifies the signed root, and hash values of the blocks received from the server.

Shao et al. [6] proposed a scheme based on the hashed multiple branches tree (HMBT). MHT is based on a binary tree, but HMBT is based on an n-ary tree. Fu et al. [7] developed a multiple hash tree (MPHT) for data integrity verification. Compared to MHT, the height of MPHT is low, which reduces the integrity verification time. Tian et al. [8] constructed a dynamic hash table for cloud data integrity verification. Shen et al. [9] proposed a method for efficient public auditing of cloud data. A location array and a doubly linked info table are the authentication structures used in this scheme. Khedr et al. [10] developed a cryptographic-accumulator model to verify the integrity of the data stored on cloud servers. The proposed model used a modified Rivest, Shamir, and Adleman-based cryptographic accumulator.

All these schemes ensure the integrity of data stored on cloud servers. Data security issues are also present in fog computing. Delivering security-as-a-service guarantees system security, including network, data, and fog node security [11]. Ahsan et al. [12] presented a technique for providing integrity and confidentiality in fog-oriented cloud storage. XOR operations and collision-resistant hash functions are used to check the integrity and retrieve corrupt data blocks. The data collected from IoT devices are encoded using Reed-Solomon code and shared between the fog and cloud storage. The hashtag associated with each block verifies integrity, and the encoding helps to recover corrupted blocks [13].

For securing road surface condition crowdsensing, Basudan et al. [14] presented a system based on certificate-less aggregate signcryption. The solution maintains the data's privacy, mutual authentication, confidentiality, and integrity. To maintain the privacy, secrecy, and integrity of vehicular crowd-sensing data, modified homomorphic encryption with a super-increasing sequence is applied [15]. Tian et al. [16] proposed a scheme to verify data integrity in fog-to-cloud storage using a tag transformation strategy. Blockchain technology is used for peer-to-peer authentication and ensuring data integrity in fog computing [17].

However, this comes at a considerable cost regarding power utilization and storage resources as the use of robust fog nodes is needed. Azeem et al. proposed a technique that protects data integrity and counters various security risks, such as data creation and replay attacks. By applying the secure message aggregation and decryption algorithm at mobile sinks and fog nodes, the proposed work provides secure data aggregation and data forwarding of healthcare data [18].

Liu et al. [19] proposed a method for securely monitoring patient healthcare data in a fog environment. Sudarsono et al. [20] suggested a method that uses CP-ABE and Hashed MAC to securely transfer environmental data from sensor nodes to a gateway while maintaining integrity. To safely encrypt the sensor data, Chatamoni et al. [21] adopted a portable compressive sensing technology that uses structurally random matrices and blocks compressive sensing.

Table 1 compares the proposed work with existing works. Most of the existing work uses a fog layer to enhance the security of data which is stored under the cloud. The data produced by IoT devices are initially stored on fog nodes. Since the fog data are accessed by the various end users, the fog storage needs to be secured. If the data stored in the fog storage are altered by malicious users, then the consumer fog nodes and the end users will get the incorrect data. It necessitates the integrity verification of fog data. The proposed work guarantees fog data integrity by using MAC, fuzzy clustering, dynamic key, and dynamic permutation table.

Table 1 Comparison of the proposed work with existing works

Author	Confidentiality	Integrity	Ensuring integrity at the storage level	Ensuring integrity at the communication level	Ensuring the correctness of fog storage	Usage of dynamic primitives
Wang et al. [5]	x	√	√	x	-	x
Shao et al. [6]	x	√	√	x	-	x
Fu et al. [7]	x	√	√	x	-	x
Tian et al. [8]	x	√	√	x	-	x
Shen et al. [9]	x	√	√	x	-	x
Ahsan et al. [12]	√	√	√	x	x	x
Wang et al. [13]	√	√	√	x	x	x
Basudan et al. [14]	√	√	x	√	x	x
Wang et al. [15]	√	√	x	√	x	x
Tian et al. [16]	x	√	√	x	x	x
Proposed	√	√	√	x	√	√

3. Proposed System

3.1. System model

Fig. 1 depicts the system model of the proposed work. The model consists of four entities: IoT devices, cloud servers, fog compute nodes (FCN), and fog storage nodes (FSN). A CSP organizes and administers many cloud servers to provide scalable and on-demand data storage. IoT devices record the raw data with the equipped sensors and send them to FCN. Fog nodes are reliable local computing centers that collect data from various IoT devices. Fog nodes, positioned between the cloud and end-user devices, can perform processing, transmission, and storage of data [22]. The proposed work considers two types of fog nodes: FCN and FSN. FCN collects data from various IoT devices, processes, and stores in FSN. The fog users can request the data through FCN. FSN stores the data processed by FCN, and each FSN is associated with the cloud server. It stores its data in its associated cloud server.

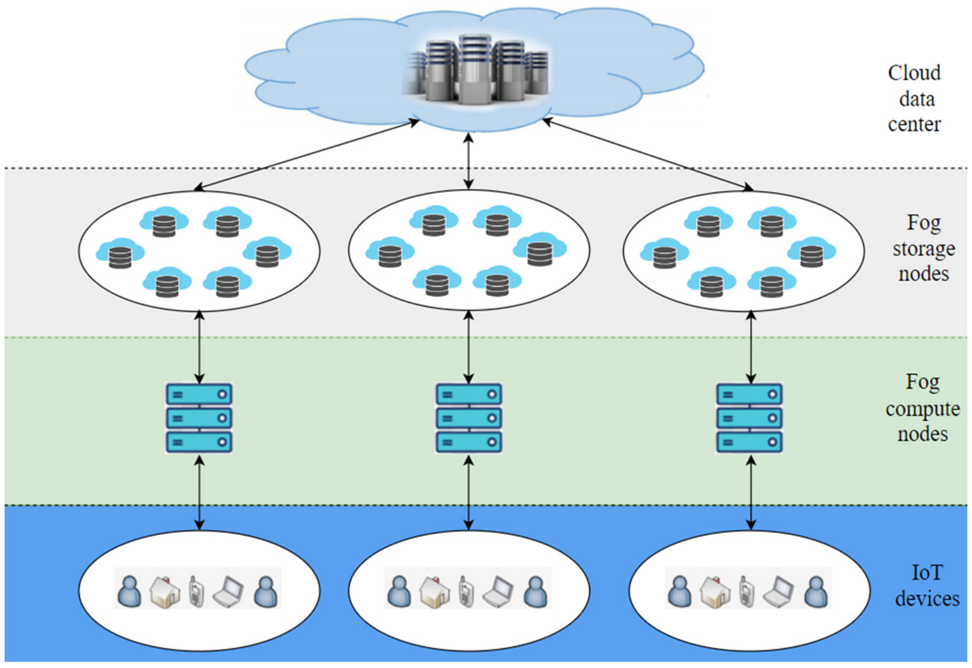

Fig. 1 System model

3.2. Preliminaries

The proposed work involves dividing the IoT data into *n* clusters using fuzzy clustering and encrypting them with a dynamic key. After that, the encrypted clusters are distributed across the *n* fog nodes by the dynamic permutation method. For data retrieval, all the *n* clusters and the appropriate dynamic key are required. From the dynamic key, the permutation box (p-box) is regenerated, and this p-box helps identify the fog storage nodes that store the corresponding clusters. Retrieved clusters are grouped and re-clustered by using fuzzy clustering to verify the integrity. Thus, the clustering technique and p-boxes play a major role in the proposed work.

3.2.1. Clustering

During the clustering process, a set of data points are divided into several groups. The data points within a group are more similar to one another than the data points in other groups. Clustering algorithms can be deterministic or non-deterministic. A deterministic algorithm always obtains the same results with the same input data. A non-deterministic algorithm may provide different results every time it runs, even if the same input values are used. When integrity verification is performed, datasets grouped in one cluster should be re-clustered. Only then will the integrity verification produce correct results. As a result, deterministic clustering is used for integrity verification in the proposed work.

The proposed work uses fuzzy clustering, which is a deterministic algorithm. Fuzzy clustering generates unique cluster centers for a given dataset. Each record of the dataset is always assigned the same membership value, which means that the records of the dataset always belong to the same cluster, regardless of the number of times that the clusters are formed. This property is helpful for data integrity verification. Metadata for integrity verification does not have to be stored locally. This saves metadata storage and maintenance overhead.

3.2.3. Dynamic permutation

Almost all traditional cryptographic algorithms such as the data encryption standard and advanced encryption standard use p-boxes. Permutations are based on a fixed table in this case. p-boxes determine the strength of cryptographic algorithms; hence, building cryptographically strong p-boxes is crucial in the design of secure cryptosystems. Based on number theory and modular computing, Zobeiri and Mazloom-Nezhad [23] suggested a method for constructing dynamic permutation tables. The length of the permutation table is not fixed in this case. The permutation table used in the proposed work is not unique, and it depends on the dynamic key. Additionally, the number of entries in the permutation table varies each time when it is constructed. These features of dynamic p-boxes make them more suitable to use in the proposed work to improve the security level.

3.3. Proposed data integrity verification approach

The cloud-based IoT network model depends on centralized cloud data centers, which is not feasible for IoT applications. As a result, fog computing was introduced to enable data manipulation at the network end to improve service quality by responding quickly to customer requests. Data stored in fog nodes are subject to attacks, just like data stored in cloud servers. As a result, the integrity of data stored in fog nodes must be verified. Various approaches for verifying data integrity have been established; however, most present techniques rely on complex cryptographic algorithms and are unsuitable for handling big data.

The proposed work uses fuzzy clustering along with dynamic permutation, dynamic key, and MAC to address this issue. The proposed approach validates data integrity by considering three phases: key generation, data processing, and integrity verification. During the key generation phase, dynamic keys are generated by FCN and used for encryption, permutation, and MAC generation. The second phase is the data processing phase where the FCN is used to encrypt, cluster the data, and distribute them to the FSN with fuzzy clustering, dynamic permutation, and a dynamic key. The third stage is the integrity verification stage, where data is decrypted, grouped, and re-clustered to verify integrity. Fig. 2 illustrates the block diagram of the proposed system model.

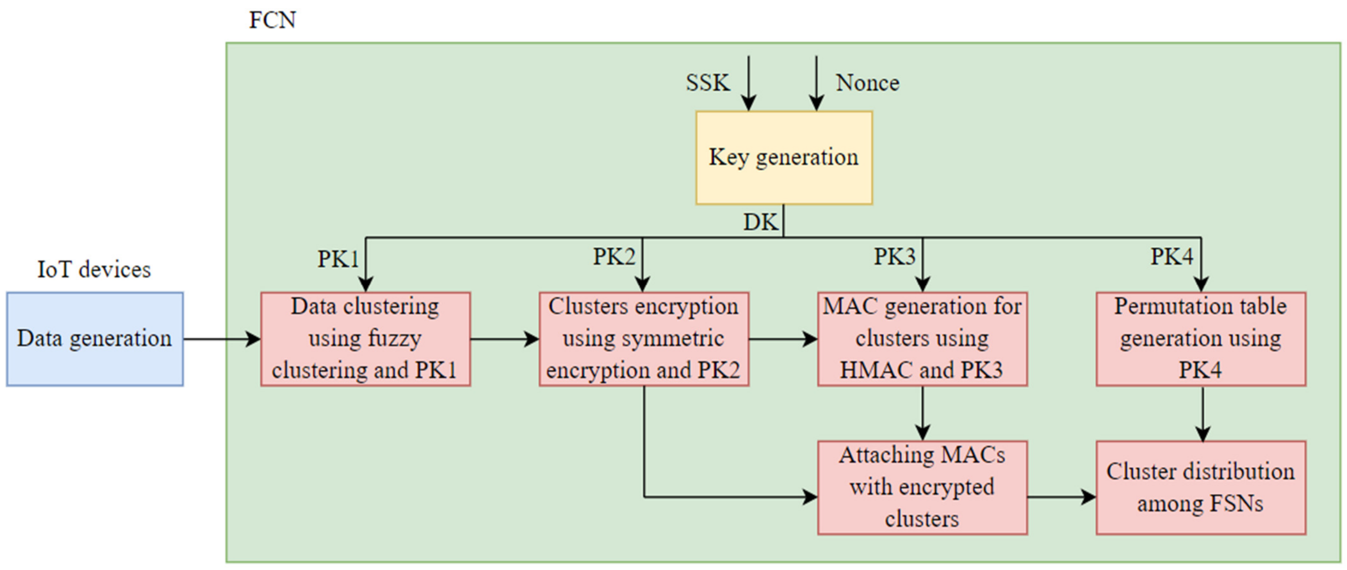
FCN
SSK
Nonce
Key generation
DK
PK1
PK2
PK3
PK4
IoT devices
Data generation
Data clustering using fuzzy clustering and PK1
Clusters encryption using symmetric encryption and PK2
MAC generation for clusters using HMAC and PK3
Permutation table generation using PK4
Attaching MACs with encrypted clusters
Cluster distribution among FSNs

Fig. 2 Block diagram of the proposed system model

3.3.1 Key generation phase

Dynamic keys and related sub-keys are generated in this phase. Fig. 3 and Fig. 4 show the generation of the dynamic key and its sub-keys. Each FCN is associated with a secure static key (SSK) that is not shared with other fog nodes. The pseudo-random number generator generates a nonce at the end of every application session. The metadata and timestamp of the application's data are used to generate a random nonce. Dynamic key (DK) is produced by XORing the nonce with the SSK and hashing it using a secure hash function. The generated DK is 512 bits long and is renewed after each session. DK is subdivided into four sub-keys that serve as seeds for various cryptographic primitives. The use of dynamic cryptographic primitives gives strong resistance to existing and new threats.

DK has a size of 64 bytes, divided into four parts, each with a length of 16 bytes. Every 16 bytes is used for different cryptographic mechanisms:

(1) Cluster sub-key PK1: The number of clusters is determined using this sub-key. It is made up of B_{63}-B_{48} bytes of DK.

(2) Encryption sub-key PK2: The encryption sub-key is used to encrypt the clusters. It is made up of DK's B_{47}-B_{32} bytes of DK.

(3) MAC generation sub-key PK3: This sub-key checks the cluster's integrity. It is made up of B_{31}-B_{16} bytes of DK.

(4) Permutation sub-key PK4: This sub-key is utilized to generate a dynamic permutation table. It is made up of B_{15}-B_0 bytes of DK.

Every session has a distinct dynamic key, and any bit change in nonce produces an entirely new set of dynamic sub-keys. As a result, different cryptographic updates and primitives are produced.

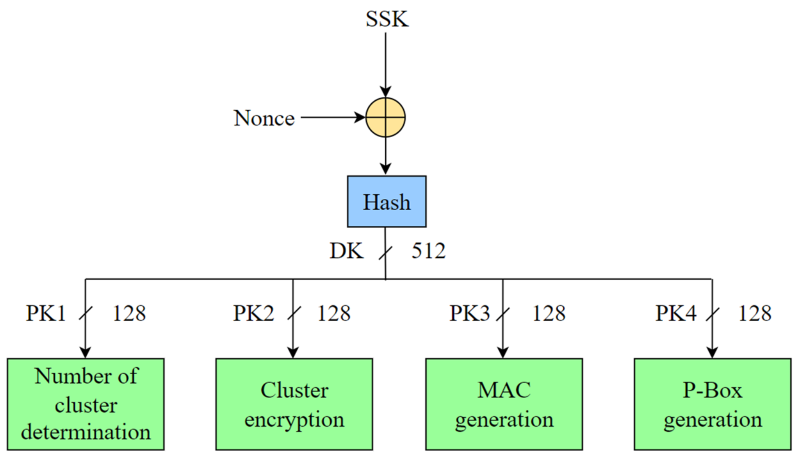

Fig. 3 Key generation phase

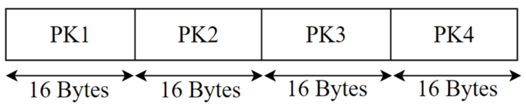

Fig. 4 Dynamic sub-keys

3.3.2 Data processing phase

The key generated during the key generation phase is divided into four 128-bit parts. The four parts are respectively used for the number of cluster selections, symmetric encryption, MAC generation, and permutation. At the end of each application session, FCN groups the data received from IoT devices and applies fuzzy clustering to the collected data.

The number of clusters to be formed is determined by PK1. After clustering, each cluster is encrypted using PK2, and a MAC is generated for each encrypted cluster using PK3. The FCN adds the MAC to each encrypted cluster and stores these clusters in FSNs using a dynamic permutation table generated with PK4.

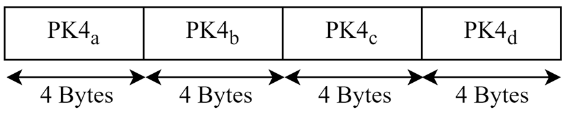

Fig. 5 Sub-keys for permutation table generation

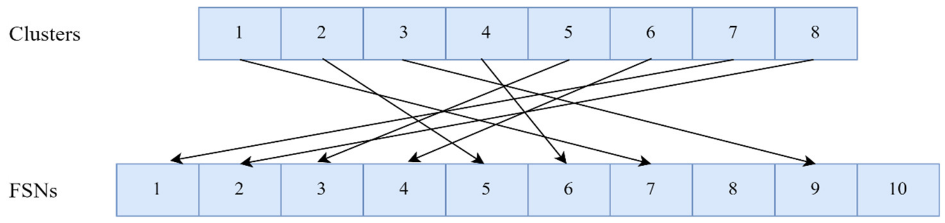

Fig. 6 Dynamic permutation with $N_C = 8$ and $N_{FSN} = 10$

To minimize communication time, only a few clusters are considered. For this, the four LSBs of PK1 or 4 bits extracted from PK1 can be used to determine the number of clusters. To generate a dynamic permutation table, arbitrary variables need to be initialized. For this, PK4 is divided into 4 sub-keys $PK4_a$, $PK4_b$, $PK4_c$, and $PK4_d$ as shown in Fig. 5. The number of entries in the input permutation table is determined by the number of clusters. The permutation table is constructed using arbitrary variables. In most cases, the entries in the input and the output permutation tables are the same. In the proposed work, the number of entries in the input table is equal to the number of clusters, whereas the number of entries in the output table is equal to the number of FSNs. Fig. 6 shows the distribution of clusters over the FSNs. The first cluster is stored in the seventh FSN, the second one is stored in the fifth FSN, and so on.

Algorithm 1 describes the steps in the data processing phase. Table 2 shows the list of notations used in algorithm 1 and algorithm 2. Initially, fuzzy clustering is applied to divide the data block into clusters. These clusters are encrypted using symmetric encryption and PK2. PK3 is used to generate the MAC for every encrypted cluster, and this MAC is appended to each encrypted cluster.

Algorithm 1: Data processing phase
Input: Dynamic sub-keys (PK2, PK3), data block (D), number of clusters (N_C)

```
1.  $C_0, C_1, …., C_{n-1} = FC(D, N_C)$
2.  for i = 0 to $N_C$ - 1 do
3.      $C_i' = Enc(PK2, C_i)$
4.      $MAC_{Ai} = MAC(PK3, C_i')$
5.      $C_i'' = MAC_{Ai} \| C_i'$
6.  end for
```

Output: MAC attached encrypted clusters

$$E_{tmp} = \left[\left(X - SP[\ ]\right) pow\left(Y + SP[\ ]\right) \% n\right] \% m \tag{1}$$

Algorithm 2: Permutation table generation
Input: Dynamic sub-keys ($PK4_a$, $PK4_b$, $PK4_c$, $PK4_d$), number of clusters (N_C), and total number of FSNs (N_{FSN})

```
1.  $PT_{ip}[\ ] = [1, 2, …., N_C]$
2.  $PT_{op}[\ ] = [\ ]$
3.  $X' = PK4_a$; $Y' = PK4_b$; $Z' = PK4_c$
4.  X = X' + 1000; Y = Y' + 1000; Z = Z' + 100
5.  Bool = 0
6.  while (Bool == 0) do
7.      if (Z is prime) then
8.          Bool = 1
9.      else Z--
10.     end If
11. end while
12. LP = Z
    //Initialization of SP array with set of prime numbers
13. SP[ ] = [LP, ….., 5, 3, 2]
14. m = n = $N_C$
```

```
15.  for k = 0 to n - 1 do
16.      $E_{tmp}$ = [(X - SP[k]) pow (Y + SP[k]) % n] % m
         //Find the $E_{tmp}^{th}$ entry in the $PT_{ip}$[] and copy it in the $PT_{op}$[]
17.      $PT_{op}[k] = PT_{ip}[E_{tmp}]$
18.      while ($E_{tmp}$ < m) do
19.          $PT_{ip}[E_{tmp}] = PT_{ip}[E_{tmp} + 1]$
20.          $E_{tmp}$++
21.      end while
22.      m = m - 1
23.  end for
24.  for z1 = 0 to $N_C$ - 1 do
25.      $PT_{op}[z1] = (PT_{op}[z1] + PK4_d) \% N_{FSN}$
26.  end for
Output: Permutation table
```

Algorithm 2 describes the process of permutation table generation. To generate a permutation table, two tables PT_{ip} and PT_{op} are used. The three variables X', Y', and Z' are initialized with $PK4_a$, $PK4_b$, and $PK4_c$. The variable LP is initialized with the largest prime number less than Z. A set of prime numbers (LP, ..., 3, 2) is used to initialize the SP array. The variables *m* and *n* are used to get the value of PT_{ip} and populate PT_{op}. The permuted entries for PT_{ip} are calculated using Eq. (1). To fill PT_{op}, Eq. (1) is repeated n times. Each time the E_{tmp} entry in the PT_{ip} is replaced by its successor, the remaining entries in the PT_{ip} are shifted to the left. To distribute clusters among available FSNs, each entry in the final permutation table is processed by $PK4_d$. The resulting permutation table is used to distribute the clusters among FSNs.

Table 2 List of notations

Notations	Meaning
D	Data block
$C_1, C_2, C_3, \ldots, C_{Nc}$	Clusters
$C_1', C_2', C_3', \ldots, C_{Nc}'$	Encrypted clusters
$C_1'', C_2'', C_3'', \ldots, C_{Nc}''$	Encrypted clusters attached with MACs
MAC_{Ai}	MAC generated from cluster C_i
N_C	Number of clusters
N_{FSN}	Number of FSNs
LP	Largest prime
SP	Set of prime numbers
PT_{ip}	Input permutation table
PT_{op}	Output permutation table
MAC(PK3, C_i')	MAC generation of C_i' by PK3
Enc(PK2, C_i)	Symmetric encryption of C_i by PK2
FC(D, N_C)	Fuzzy clustering of D with N_C

3.3.3. Integrity verification phase

In the study, integrity verification is completed on two levels: cluster level and block level. When the user requests the data, FCN derives DK from the stored SSK and nonce. FCN computes the number of clusters using PK1 of DK and finds the associated FSNs using PK4 of DK. It gets the clusters from the relevant FSNs and uses PK3 to generate MACs for every cluster. The cluster's integrity is guaranteed if a newly generated MAC matches a previously stored MAC. Hence, the integrity is verified at the cluster level. The clusters are then decrypted using PK2 and a symmetric decryption algorithm. FCN combines the clusters into a single data block before re-clustering them. Records belonging to the same cluster should have the same membership values. This attribute aids in the verification of the received data block's integrity. The data block is re-clustered using a fuzzy clustering algorithm, and membership values are examined to verify the integrity at the block level.

Clustering is an efficient method for identifying patterns and structures in labeled and unlabeled datasets [24]. Most data security methods in fog computing collect raw IoT data, apply encryption mechanisms and store it on a server. It is far more beneficial to secure processed data than raw data. In the proposed work, fuzzy clustering is used to find useful patterns in the dataset and helps to ensure the integrity of the data stored in the fog layer. Then cryptographic mechanism is applied to the clusters to store them securely in the FSNs. Fig. 7 shows the integrity verification phase of the proposed approach.

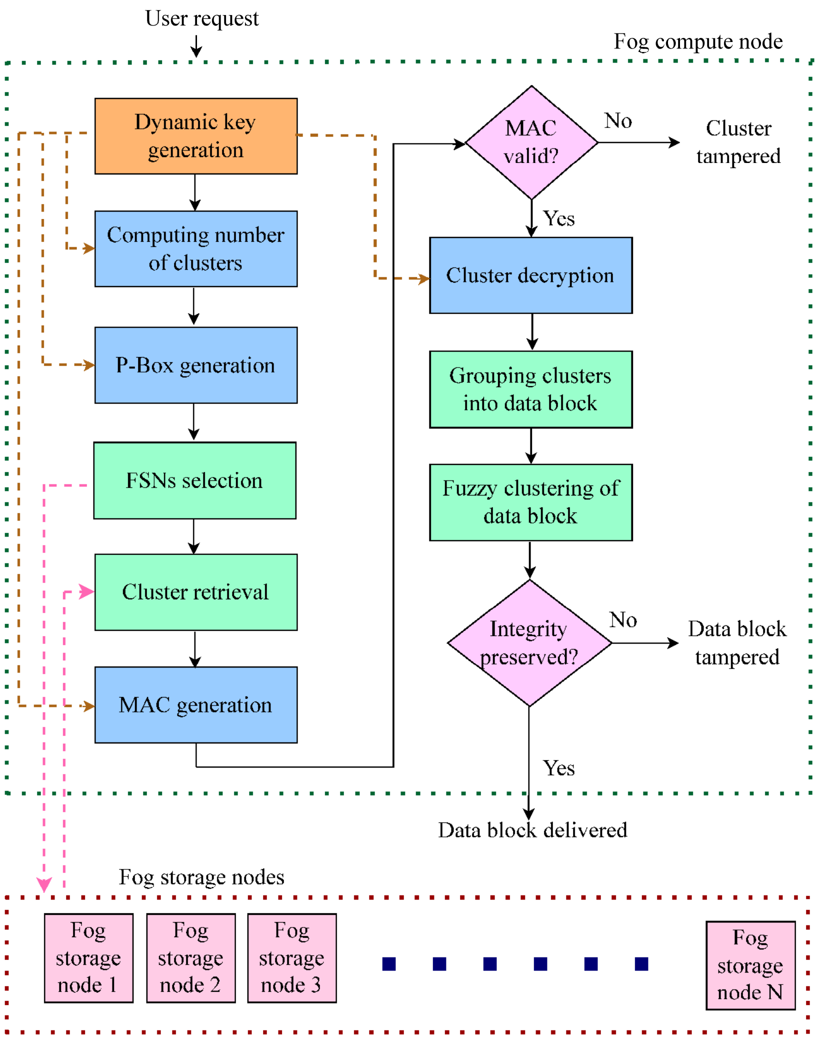

Fig. 7 Integrity verification phase

The dataset Zaxb was partitioned into three shares, Z1pxb, Z2qxb, and Z3rxb, and then the three shares were distributed into 3 FSNs. The integrity of original data can be verified only when all shares are received. In fuzzy clustering, every object can belong to every cluster with a membership weight between zero and one. If the weight is one, then the object belongs to that cluster. Fuzzy clustering works as follows:

(1) Initially, the number of clusters N_C is determined, and N_C cluster centers are selected randomly from the dataset.

(2) Membership probability for each data point is calculated as follows:

$$U_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{D_{ij}}{D_{ik}} \right)^{\frac{2}{f-1}}} \tag{2}$$

where D_{ij} is the distance between the ith data point to the jth cluster center, and f is a fuzziness parameter, ranging from 1 to infinity. The fuzziness parameter establishes the degree of data point sharing between clusters. In the proposed work, it is set to 1.

(3) Cluster centers are calculated by:

$$CC_{jk} = \frac{\sum_{i=1}^{N} U_{ij}^{f} x_{ik}}{\sum_{i=1}^{N} U_{ij}^{f}} \tag{3}$$

where x_{ik} is the data point.

(4) Steps 2 and 3 are repeated till the cluster centers remain constant.

This procedure is used for integrity verification at the block level. During integrity verification, clusters are retrieved from the FSN and grouped, cluster centers are randomly selected, and the membership values are calculated for each record. Cluster centers are recalculated until they stop changing. Once the cluster centers remain unchanged, the membership values of each record are checked to verify the integrity of the retrieved data.

4. Security Analysis and Performance Analysis

The proposed work uses a dynamic key approach, which generates a set of highly random dynamic sub-keys. All cryptographic operations are associated with dynamic keys that ensure the desired cryptographic performance. This section describes the security and performance analysis of the proposed work.

4.1. Security analysis

This section analyses the security of the proposed work by considering various security attacks. Key sensitivity analysis shows that small changes in dynamic keys lead to drastic changes in the permutation table. The cryptanalysis discussion addresses the difficulty of attackers in recovering the original data. The proposed approach is resilient to the following attacks:

(1) Linear and differential cryptanalysis attack: Linear cryptanalysis is a type of cryptanalysis attack that looks for a relationship between the input and output of a cipher scheme. Differential cryptanalysis attack exposes a different pattern between two plaintexts and their associated ciphertexts.

(2) Chosen/known plaintext/ciphertext attack: A cryptanalyst can choose any plaintext/ciphertext and then obtains the associated ciphertext/plaintext. Thus, attackers try to either obtain the secret encryption key or devise an algorithm that would allow decrypting all ciphertext messages encrypted with this key.

(3) Session key attack: Attackers obtain the session key for a particular session and utilize it for other sessions.

(4) Eavesdropping attack: It captures data sent over a network by a computer or other connected devices. Attackers can monitor and listen in on network communications.

Owing to the use of a dynamic key and an efficient permutation mechanism, the proposed approach is secure against all the attacks mentioned above. Furthermore, even if an adversary successfully recovers one generation, all subsequent generations cannot be recovered because the key is updated dynamically.

4.1.1 Brute-force attack

Brute force attacks are most often used to compromise keys. This can be achieved by trying all key possibilities using known/selected plaintext/ciphertext pairs to obtain the desired result. In this case, the length of the key is critical to protect the attacker's work. The secret key should be at least 128 bits long, which corresponds to a key space of 2^{128} bits. In the proposed approach, the dynamic key is 512 bits in length, with a key space of 2^{512} corresponding to it, which is sufficient to make the brute-force attack infeasible.

The time required to crack a key grows exponentially with the length of the key. In the proposed work, dynamic keys significantly complicate the attackers' work. Attackers must know the key during the session to reveal the original data. Knowing the key for a single session does not guarantee that an attacker can access data processed in the past or the future.

4.1.2. Key sensitivity

The key sensitivity test modifies the PK4 slightly and computes the difference between the permuted entries at the bit level. A key sensitivity value close to 50% should be guaranteed by an effective cryptographic system. Two secret keys Key1 and Key2 are used in this test; both of them are identical, except for one random bit. The corresponding p-boxes are computed, and the hamming distance between these two p-boxes is calculated as follows:

$$KS = \left(\frac{\sum_{k=1}^{T} C_1 \oplus C_2}{T} \right) 100\% \tag{4}$$

where T is the bit-level permutation table's length. The key sensitivity test is shown in Fig. 8 for 1,000 iterations; the mean result is close to 50% with a low standard deviation. The proposed work can therefore guarantee high resistance to related-key attacks and linear and differential attacks.

4.1.3. Cryptanalysis discussion

The proposed work's cryptographic security is based on using variable cryptographic primitives. The previous or next set of encrypted clusters cannot be recovered even with complete knowledge of the encrypted cluster since new cryptographic primitives are applied for each application session. This ensures both forward and backward secrecy. Additionally, the permutation method is used to strengthen the unpredictability of the clusters before the data is stored in FSNs. The proposed work distributes encrypted clusters over various FSNs. Therefore, the original data protection relies on the task of extracting the clusters from N_{FSN} FSNs, as well as the challenge of finding the proper session key, which is sufficiently long to make it impossible.

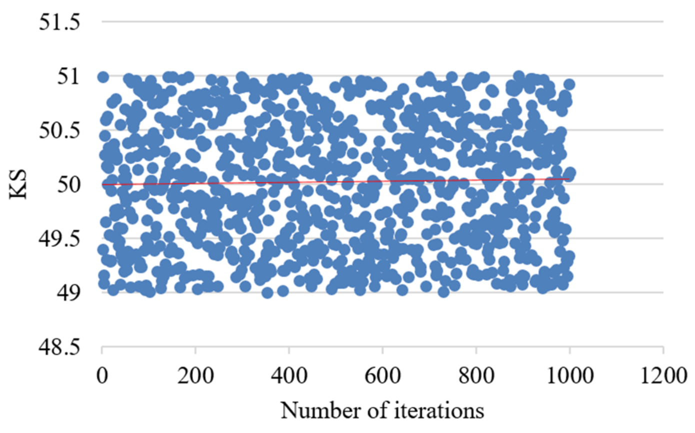

Fig. 8 Key sensitivity variations

4.2. Performance analysis

This section compares the proposed work based on fuzzy clustering with other clustering algorithms. The subsections analyze the proposed work in terms of computation, communication, and storage overhead. The proposed work was deployed and tested on Raspberry Pi. In the experimental setup, Raspberry Pi represents the FCN. The Raspberry Pi kit used in this experiment is RPI4. RPI4 has a 1.5GHz quad-core microprocessor with ARMv7 instructions and 8GB RAM. For analysis, the computation time of the proposed work is compared against k-means, agglomerative clustering with three and five as the hyperparameter value of the number of clusters. Since most of the computations take place in FCN, the performance of the FCN has been analyzed with data integrity verification time.

The fuzzy clustering algorithm is used for forming the clusters, AES-128 symmetric encryption algorithm is used for encrypting the clusters, SHA-512 is used for dynamic key generation, and HMAC is used for generating the MAC value. The efficiency of the proposed partitioning-based data sharing approach using computation time is adapted for assessment. The assessment is performed with the mHealth dataset [25] developed by the University of Granada, Spain. It includes capturing the vital signs and body motions of ten different participants. Sensor readings are obtained from the volunteer's left ankle, right wrist, and chest. The sensor readings are also treated as healthcare data obtained from IoT devices. Fig. 9 and Fig. 10 show the integrity verification time of data blocks with varying data sizes from 1MB to 5MB using fuzzy clustering, K-means clustering, and agglomerative clustering.

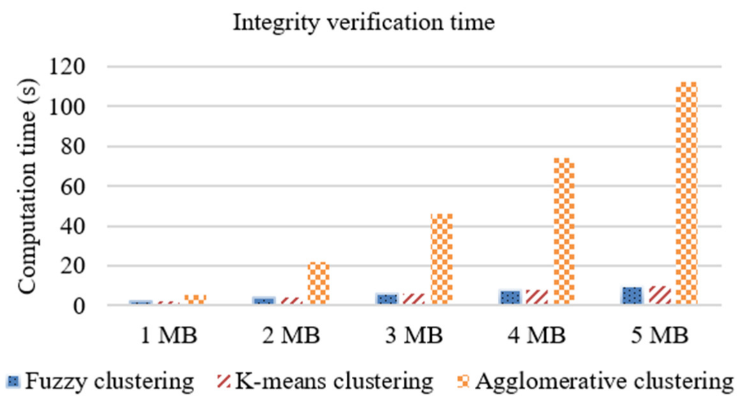

Fig. 9 Integrity verification time with $N_C = 3$

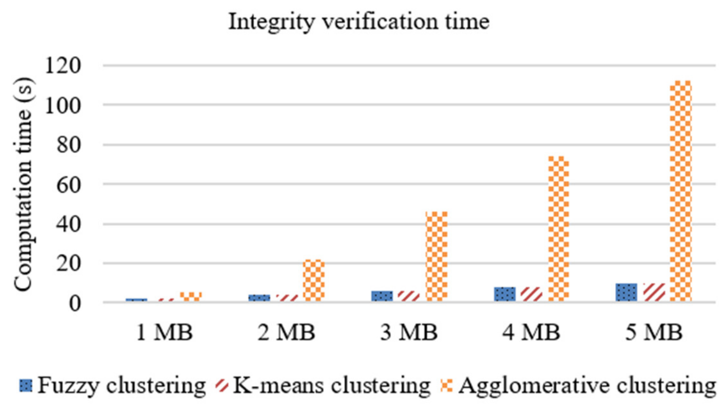

Fig. 10 Integrity verification time with $N_C = 5$

From the analysis, it is identified that fuzzy clustering and K-means clustering took minimal computation time compared to agglomerative clustering. But K-means clustering is a non-deterministic algorithm. This means that using the algorithm multiple times on the same set of data may provide different outcomes, as shown in Fig. 11. In addition, K-means do not produce the best results and only works when the clusters are of diverse sizes and densities. If the initial centers are chosen incorrectly, the solution may focus on local maxima rather than global maxima. These characteristics make the K-means clustering to be unsuitable for integrity verification. Likewise, agglomerative clustering consumes higher computation time than the other two clustering techniques. Hence, fuzzy clustering is a suitable technique for integrity verification in the fog layer.

Fuzzy clustering uses a deterministic approach and gives each record a membership value based on its distance associated with the cluster centers. Records are clustered together using these membership values. Fuzzy clustering always generates the same cluster centers for a given dataset. It consistently assigns the same membership value to each record in a dataset, meaning that records always belong to the same cluster, regardless of the number of executions. This characteristic aids in the verification of data integrity. Fig. 12 shows the distribution of records among several clusters. The membership values assigned to each record during clustering are not required to be maintained in local storage. Hence, the proposed approach is efficient in terms of memory utilization.

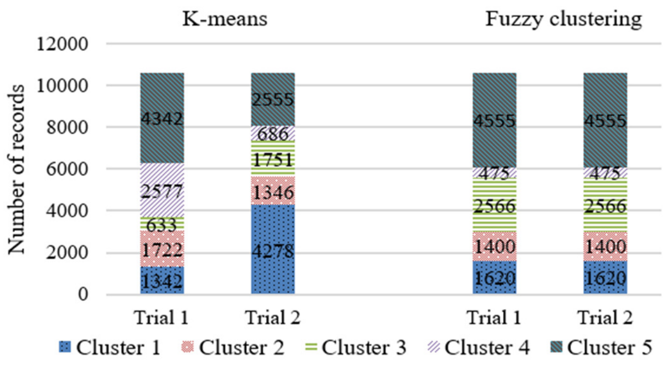

Fig. 11 Cluster formation in K-means vs fuzzy clustering

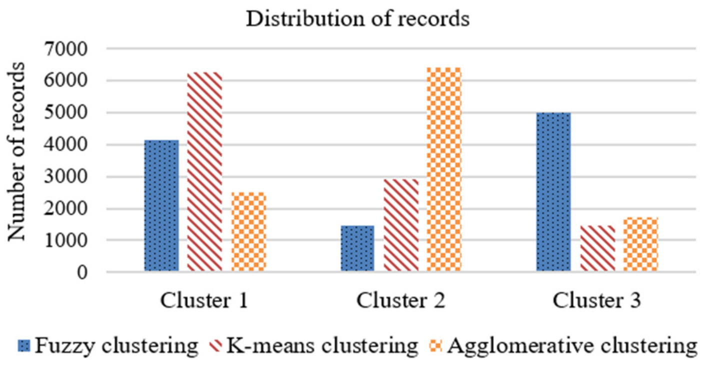

Fig. 12 Distribution of records in different clustering techniques with $N_C = 3$

4.2.1 *Computation time*

The computation time involves generating dynamic keys using SHA-512 and XOR operations, encrypting data using the AES algorithm, generating MAC using the HMAC algorithm, partitioning data using fuzzy clustering, and generating a permutation table using modular exponential operations. The initial data upload time is $T_{SHA} + T_{XOR} + T_{AES} + T_{HMAC} + T_{FC} + T_{PT}$. The data integrity check time involves the same operations, except AES decryption. In addition, the time to group the clusters into a block and the final verification of the membership value of the newly formed clusters are also considered to calculate the computation time. Fig. 13 shows that the proposed scheme performs 50 times better than the tag regeneration scheme [16]. The proposed scheme mainly involves a few modular exponential operations, XOR operations, and hashing operations, while the tag regeneration scheme mainly involves bilinear pairing operations, exponential operations, and hashing operations.

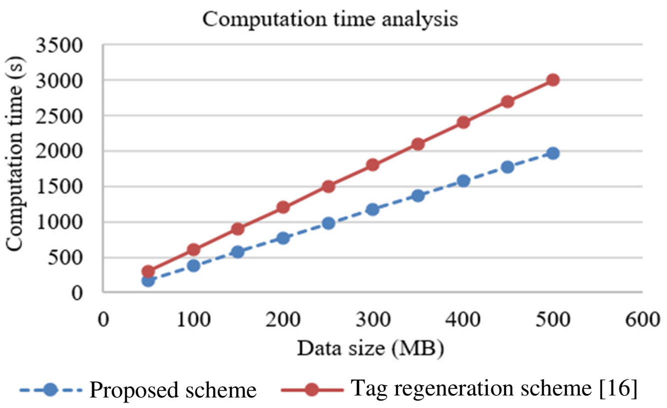

Fig. 13 Comparison of the proposed work with the tag regeneration scheme

4.2.2 Communication time

The cluster size and the number of clusters to be verified influence the communication time in the proposed work. Communication time increases with the number of clusters. However, the proposed work greatly reduces communication time owing to the adoption of the fog layer. Fog computing reduces the amount of data that goes to the cloud, significantly reducing the latency, response time, and end-to-end latency of data transmission. Several schemes use a fog layer to process IoT data and store the processed data in cloud storage, while the proposed work stores data in fog storage and serves end users [12-13, 16]. Therefore, the proposed work reduces the overall communication time.

4.2.3 Storage overhead

Regarding storage overhead, FCN uses SSK and a nonce to generate a dynamic key at the end of each application session. Regardless of the number of connected IoT devices, FCN uses a single 512-bit nonce to generate dynamic keys and process data received from the IoT layer. During data retrieval, FCN regenerates dynamic keys using the same nonce. Hence, the storage overhead for integrity verification on the FCN side of a single session is only 512 bits. During the data processing phase, a 128-bit MAC is attached to each cluster, which is then checked during data retrieval. Therefore, the storage overhead of FSN is 128 bits per cluster.

5. Conclusions

In this study, a partitioning-based data integrity verification approach is proposed. The network contains four entities: IoT nodes, FCNs, FSNs, and cloud servers. The procedure of this approach includes three phases protecting and maintaining the integrity of the data stored in the fog layer. These phases are the key generation phase, the data processing phase, and the integrity verification phase. During the key generation phase, a secure dynamic key is generated, and sub-keys are derived from it. In the data processing phase, the received IoT data are clustered and encrypted, and MAC tags are generated for each cluster. To store the clusters in distributed FSNs, a dynamic p-box is generated, and the clusters are distributed. Finally, in the integrity verification phase, the steps of the data processing phase are performed in reverse order to verify the integrity of the data.

The proposed partitioning-based sharing approach provides improved performance for the highest level of security. This approach focuses on verifying the integrity of a single data replica stored on fog nodes. In a large-scale fog infrastructure, multiple data replicas are placed on various fog nodes to minimize the access latency of data consumers. In the future, the proposed work can be extended to verify the integrity of the multiple data replicas in large-scale fog infrastructures. Moreover, clustering can be coupled with deep learning to improve the efficiency of data integrity verification.

Conflicts of Interest

The authors declare no conflicts of interest.

References

[1] C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-Preserving Public Auditing for Secure Cloud Storage," IEEE Transactions on Computers, vol. 62, no. 2, pp. 362-375, February 2013.

[2] X. Zhang and W. Si, "Efficient Auditing Scheme for Secure Data Storage in Fog-to-Cloud Computing," IEEE Access, vol. 9, pp. 37951-37960, 2021.

[3] H. Noura, O. Salman, A. Chehab, and R. Couturier, "Preserving Data Security in Distributed Fog Computing," Ad Hoc Networks, vol. 94, article no. 101937, November 2019.

[4] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, et al., "Provable Data Possession at Untrusted Stores," Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 598-609, October 2007.

[5] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, “Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing,” IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 5, pp. 847-859, May 2011.

[6] B. Shao, G. Bian, Y. Wang, S. Su, and C. Guo, “Dynamic Data Integrity Auditing Method Supporting Privacy Protection in Vehicular Cloud Environment,” IEEE Access, vol. 6, pp. 43785-43797, 2018.

[7] A. Fu, Y. Li, S. Yu, Y. Yu, and G. Zhang, “DIPOR: An IDA-Based Dynamic Proof of Retrievability Scheme for Cloud Storage Systems,” Journal of Network and Computer Applications, vol. 104, pp. 97-106, February 2018.

[8] H. Tian, Y. Chen, C. C. Chang, H. Jiang, Y. Huang, Y. Chen, et al., “Dynamic-Hash-Table Based Public Auditing for Secure Cloud Storage,” IEEE Transactions on Services Computing, vol. 10, no. 5, pp. 701-714, September-October 2017.

[9] J. Shen, J. Shen, X. Chen, X. Huang, and W. Susilo, “An Efficient Public Auditing Protocol with Novel Dynamic Structure for Cloud Data,” IEEE Transactions on Information Forensics and Security, vol. 12, no. 10, pp. 2402-2415, October 2017.

[10] W. I. Khedr, H. M. Khater, and E. R. Mohamed, “Cryptographic Accumulator-Based Scheme for Critical Data Integrity Verification in Cloud Storage,” IEEE Access, vol. 7, pp. 65635-65651, 2019.

[11] Y. Guan, J. Shao, G. Wei, and M. Xie, “Data Security and Privacy in Fog Computing,” IEEE Network, vol. 32, no. 5, pp. 106-111, September-October 2018.

[12] M. A. M. Ahsan, I. Ali, M. Imran, M. Y. I. B. Idris, S. Khan, and A. Khan, “A Fog-Centric Secure Cloud Storage Scheme,” IEEE Transactions on Sustainable Computing, vol. 7, no. 2, pp. 250-262, April-June 2022.

[13] T. Wang, J. Zhou, M. Huang, M. Z. A. Bhuiyan, A. Liu, W. Xu, et al., “Fog-Based Storage Technology to Fight with Cyber Threat,” Future Generation Computer Systems, vol. 83, pp. 208-218, June 2018.

[14] S. Basudan, X. Lin, and K. Sankaranarayanan, “A Privacy-Preserving Vehicular Crowdsensing-Based Road Surface Condition Monitoring System Using Fog Computing,” IEEE Internet of Things Journal, vol. 4, no. 3, pp. 772-782, June 2017.

[15] B. Wang, Z. Chang, Z. Zhou, and T. Ristaniemi, “Reliable and Privacy-Preserving Task Recomposition for Crowdsensing in Vehicular Fog Computing,” IEEE 87th Vehicular Technology Conference (VTC Spring), pp. 1-6, June 2018.

[16] H. Tian, F. Nan, C. C. Chang, Y. Huang, J. Lu, and Y. Du, “Privacy-Preserving Public Auditing for Secure Data Storage in Fog-to-Cloud Computing,” Journal of Network and Computer Applications, vol. 127, pp. 59-69, February 2019.

[17] R. Almadhoun, M. Kadadha, M. Alhemeiri, M. Alshehhi, and K. Salah, “A User Authentication Scheme of IoT Devices Using Blockchain-Enabled Fog Nodes,” IEEE/ACS 15th International Conference on Computer Systems and Applications, pp. 1-8, October-November 2018.

[18] M. Azeem, A. Ullah, H. Ashraf, N. Jhanjhi, M. Humayun, S. Aljahdali, et al., “FoG-Oriented Secure and Lightweight Data Aggregation in IoMT,” IEEE Access, vol. 9, pp. 111072-111082, 2021.

[19] X. Liu, R. H. Deng, Y. Yang, H. N. Tran, and S. Zhong, “Hybrid Privacy-Preserving Clinical Decision Support System in Fog–Cloud Computing,” Future Generation Computer Systems, vol. 78, no. 2, pp. 825-837, January 2018.

[20] A. Sudarsono, S. Huda, N. Fahmi, M. U. H. Al-Rasyid, and P. Kristalina, “Secure Data Exchange in Environmental Health Monitoring System Through Wireless Sensor Network,” International Journal of Engineering and Technology Innovation, vol. 6, no. 2, pp. 103-122, April 2016.

[21] A. K. Chatamoni and R. N. Bhukya, “Lightweight Compressive Sensing for Joint Compression and Encryption of Sensor Data,” International Journal of Engineering and Technology Innovation, vol. 12, no. 2, pp. 167-181, February 2022.

[22] B. A. Martin, F. Michaud, D. Banks, A. Mosenia, R. Zolfonoon, S. Irwan, et al., “OpenFog Security Requirements and Approaches,” IEEE Fog World Congress, pp. 1-6, October-November 2017.

[23] M. Zobeiri and B. M. N. Maybodi, “Introducing Dynamic P-Box and S-Box Based on Modular Calculation and Key Encryption for Adding to Current Cryptographic Systems Against the Linear and Differential Cryptanalysis,” ARPN Journal of Engineering and Applied Sciences, vol. 12, no. 3, pp. 856-862, February 2017.

[24] H. Alashwal, M. El Halaby, J. J. Crouse, A. Abdalla, and A. A. Moustafa, “The Application of Unsupervised Clustering Methods to Alzheimer's Disease,” Frontiers in Computational Neuroscience, vol. 13, pp. 92-100, May 2019.

[25] “UCI Machine Learning Repository,” http://archive.ics.uci.edu/ml/datasets/mhealth+dataset, July 30, 2022.

Copyright© by the authors. Licensee TAETI, Taiwan. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY-NC) license (https://creativecommons.org/licenses/by-nc/4.0/).