Research Article

FogDedupe: A Fog-Centric Deduplication Approach Using Multi-Key Homomorphic Encryption Technique

Mohamed Sirajudeen Yoosuf, C. Muralidharan, S. Shitharth, Mohammed Alghamdi, Mohammed Maray, and Osama Bassam J. Rabie

1School of Computer Science and Engineering, VIT-AP University Amaravati, Andhra Pradesh, India
2School of Computing Technology, SRM University, Chennai, India
3Department of Computer Science and Engineering, Kebri Dehar, Kebri Debar University, Ethiopia
4Department of Information Systems, College of Computer Science, King Khalid University, Abha, Saudi Arabia
5Department of Information and Technology Systems, College of Computer Science and Engineering, University of Jeddah, Jeddah, Saudi Arabia
6Department of Information Systems, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia

Correspondence should be addressed to S. Shitharth; shitharths@kdu.edu.et

Received 12 July 2022; Accepted 9 August 2022; Published 25 August 2022

1. Introduction

Cloud computing is a technological revolution that has enabled service providers to deliver computing resources to their users through the internet. It provides easy, scalable access to the applications installed and managed in the cloud servers. The usage of cloud computing services is increasing exponentially [1]. According to the market research report from Cision 2020, the market size of cloud computing is expected to reach $832.1 billion by 2025 with a CAGR of 17.5%. On the other hand, the advancements in communication technologies and the increase in the usage of IoT devices have resulted in increased data growth. On average, 2.5 quintillion bytes of data are generated every day. As per
the reports of International Data Corporation 2020, it is expected that in 2025, around 80 Zettabytes of data might be generated from IoT and smart devices [2].

Storing and managing a large quantity of data in the cloud storage servers degrades the performance of the applications that run on the cloud. It is important to address the performance degradation issues in cloud computing technology as the data are expected to grow exponentially in near future.

Storing multiple copies of the same data in the cloud storage servers is one of the reasons for performance degradation in cloud services. In 2017, Waterford Technologies, UK claimed that around 80% of data stored in the public cloud by corporate companies is redundant. Also, the MNC companies are incurring a 12% of revenue loss every year for storing redundant data in the cloud [3]. To address the performance degradation and to remove the redundant data from the cloud storage servers, deduplication schemes are executed. It identifies redundant data and eliminates it from cloud storage servers. Many cloud providers have implemented data deduplication algorithms in their cloud architecture to improve their performance. Data deduplication schemes ensure that only one copy of the data is stored in the cloud storage server. It identifies the redundant data and replaces them with a pointer to the original copy.

The deduplication techniques can be categorized into two types based on the location where the deduplication algorithm is executed. The first type is source-level deduplication where the redundancy check is performed before the data enters the cloud storage. It decreases the ingest rates of real-world data. The second type of deduplication is post-progress deduplication, in which the redundancy check is performed only after the data enters the cloud storage. To execute the post-progress deduplication algorithm, the cloud service provider must have enough space to store the full backup (unique as well as the duplicate copies) somewhere until the duplicate data is removed from the cloud storage servers.

Existing source-level deduplication schemes Hur et al. [4], Patgiri et al. [5], and Chhabraa et al. [6] use Bloom filter as an index table in the cloud servers to perform redundancy checks. Bloom filter is a space-efficient probabilistic data structure that helps in searching for a particular element from a large set. The applications of the Bloom filter in cloud computing are keyword search, retrieving the documents from cloud storage, and cache memory. However, the Bloom filter-based index tables cannot be implemented directly on the deduplication scheme as it has a high possibility of false-positive errors. Also, existing post-progress deduplication schemes Li et al. [7], Zhou et al. [8], Liu et al. [9], Liu et al. [10], and Shen et al. [11] use convergent key encryption methods to perform deduplication, in which the data blocks are hashed and the private keys for the encryption are derived from the message digest. However, the deterministic property of the hash function may produce the same private key and identical cipher text for the redundant data blocks. Later, by comparing the identical ciphertext, the redundant data are removed from the cloud storage servers. The convergent key-based encryption method is easy and efficient to perform post-progress deduplication. However, the convergent key-based encryption methods are highly vulnerable to confirmation of file attacks (CFA) and increase privacy and security issues. Also, the convergent key-based deduplication schemes become inefficient when the number of data blocks is high.

To reduce the bandwidth wastage in the source-level deduplication, and CFA security issues in the post-progress deduplication, a FogDedupe framework is proposed. Instead of performing source-level deduplication in the cloud servers, the proposed framework introduces a concept of fog-centric deduplication, which effectively reduces bandwidth wastage. Also, to perform post-progress deduplication, additive homomorphic encryption is proposed. The data owners use a multi-key homomorphic encryption algorithm to secure their data and that allows the cloud administrator to perform operations on the corresponding ciphertexts without compromising the security. The proposed multi-key homomorphic deduplication technique allows the DOs to use different private keys to encrypt the data.

1.1. Drawbacks of Existing Deduplication Schemes. The following drawbacks in the existing deduplication schemes have to be addressed to improve the security and performance of cloud services.

(i) The existing convergent key-based encryption model has a high probability of information leakage as it is vulnerable to confirmation of file attacks (CFA)

(ii) High probability of false-positive issues in source-level deduplication when the incoming data increases

(iii) Wastage of network bandwidth in source-level deduplication, i.e., the redundancy check is performed only at the premises of the CSP. So, the redundant data and its attributes are transferred to the cloud server and that increases the communication overhead

1.2. Contributions. Our research work proposes a FogDedupe framework that executes source-level deduplication on the fog layer and post-progress deduplication on the cloud storage server. The contributions of the proposed FogDedupe frameworks are as follows:

(i) FogDedupe framework implements both source-level and post-progress deduplication simultaneously to increase the performance of the cloud services

(ii) The source-level deduplication is performed on the fog nodes which are placed near the cloud customers. It efficiently reduces bandwidth wastages

(iii) To perform source-level deduplication, a distributed index table (DIT) is created based on the Bloom filter and master-slave protocol
reduce network traffic, which is unavoidable in cloud storage system management. To perform a redundancy check using its hash values, the cloud users might end up using a lot of bandwidth usage in large numbers. Only after performing an application is mainly used to reduce the network traffic.

2.1. Source Deduplication Technique.

Deduplication techniques as source-level and target-level deduplication. 

2. Related Works

For efficient storage utilization, many cloud service providers (CSP) such as IBM Cloud, Dropbox, Amazon Web Service, and Google Drive are using deduplication techniques in the cloud environment. This section explains the recent research works which are related to performing deduplication on cloud storage servers. There are two types of deduplication techniques as source-level and target-level deduplication.

2.1. Source Deduplication Technique. The source-level deduplication is mainly used to reduce the network traffic and bandwidth usage in large numbers. Only after performing a redundancy check using its hash values, the cloud users are allowed to transmit data blocks to cloud servers. Therefore, source deduplication has become very popular and unavoidable in cloud storage system management. To reduce network traffic, the popular cloud service providers (CSP) such as Wuala, Mozy, and Dropbox are using source-level deduplication. Some of the most popular source-level deduplication techniques are Veritas Symantec NetBackup and Amazon CommonVault.

Halevi et al. [12] have proposed source-level deduplication in the cloud storage system. Here, the data owner has to compute the hash value for each data block and sends it to the cloud server whenever the user wants to upload the data to the cloud storage. A hash table is maintained by the cloud server to store all the received data blocks and performs a redundancy check for the newly received data blocks. If there is no match found in the hash table, then the data blocks will be allowed to enter the cloud storage server. Else means that the data block is redundant. Here, the uses of hash values are as follows: (1) verify the redundancy of the data block by the cloud server and (2) act as a “proof of owner” (PoW) to the data owner. If the attacker intentionally or accidentally gains access to the hash value of the data blocks, then the attacker may claim ownership of the particular data. Internal adversarial attacks are possible as the cloud server maintains the hash value of all the user’s data blocks. The scalability feature is not used in a traditional hashing table method. The hash collision rate of the hash table will be increased when the user/data block increases, and it will provide erroneous (false positive) redundant results.

To overcome the hash-based proof of ownership security threat in Halevi’s source-level deduplication, Pietro et al. [13], have proposed an s-PoW (secure PoW) method which is based on a challenge-response scheme. Here, to prove the ownership of the data, the server challenges the cloud users, and the data owner responds with some particular bits of the requested file. This method fails to address the security threats related to internal adversary attacks and to support scalability. Blasco et al. [14] have introduced a PoW verification scheme based on the Bloom filter called “bf-PoW.” It is more efficient than Halevi’s method and Pietro’s method. But this method is not assuring scalability in handling a very large volume of user data.

Zhong et al. [15] have implemented a convergent key-based proof of ownership in the cloud storage system which follows Douceur et al. [16]. To verify the ownership, the cloud server uses a convergent key instead of using hash values as PoW, which is created from the hash values of the data block, a master key is used to encrypt the actual data blocks. In this method, two keys were used to protect the user data a convergent key (to verify PoW), and the other is the master key to encrypt the data blocks. This method used both convergent keys and master keys which are created by the cloud server. Here, internal adversarial attacks are possible.

Agarwala et al. [17] have implemented source-level deduplication for images using the DICE (dual integrity convergent key) protocol. In this method, message locked encryption is used to encrypt the images. This method implemented the DICE protocol on each data block instead of encrypting the image (message) as a single file; it is decomposed into several data blocks. Here, the common blocks between two or more images are stored only once at the cloud storage. Youn et al. [18] have introduced a variant of source-deduplication using CP-ABE (Cipher policy attribute-based encryption) [13] where authorized convergent encryption is formed from attribute-based encryption (ABE). It allows only authorized users to access data stored in the cloud. Both use a third-party authorization server to generate keys for the cloud users. Yoosuf et al. [19] proposed a dual auditing scheme and an inline deduplication scheme using Bloom filters.

2.2. Post-Progress Deduplication. Post-progress deduplication is introduced to reduce the workload on cloud users because the source-level deduplication gives an extra workload (hashing data blocks, communicating hash values, and ownership tags to cloud server) to the cloud users. Here, a cloud user is unaware of the deduplication process which is performed to attain maximum storage efficiency. The workload on the client-side in performing target-level deduplication is nil. Here, the storage efficiency is improved by target deduplication.

Bellari et al. [20, 21] have introduced the first target deduplication method called DupLESS architecture using the message locked encryption (MLE) technique. The
private key is generated based on the data file (message) from the dedicated key-server which is received by the cloud users. A unique key is created by the MLE key generation algorithm for each message based on the content of the data. This key is used for data file encryption and mapping with a particular tag “T.” These tags are used for the file redundancy check, and the deduplication is performed on the storage server. The fixed and shorter keys are generated by the key server which avoids extra storage overhead. Bellari’s DupLESS architecture fails to address the solution for internal adversarial attacks because the keys are generated by the key server (cloud key server or third-party key server) from the content of data (message), which leads to the possibility of internal adversarial attacks. It also fails to support block-level deduplication and lacks security against brute force attacks.[22, 23].

Chen et al. [24] have modified the Bellari et al.’s [21] method and proposed a BL-MLE (block level-message locked encryption) to perform block-level deduplication for larger files in the cloud storage. This method addresses the issues which are block key management and proof of ownership in Bellari’s method. In the BL-MLE method, for any given input file, a master key, a single file tag, and a set of block-level keys are generated. These file tags and block tags are used to perform deduplication on the cloud storage system. Like the MLE method, BL-MLE also has a third-party key server, which creates a path to internal adversarial attacks.

Li et al. [7] have implemented a modified convergent key-based target deduplication, where the cloud user used a master key to encrypt the convergent key which is generated by the cloud server. The encrypted convergent keys are stored in the cloud storage. This modified technique uses a master-convergent key approach where an enormous number of keys are generated when the data blocks increase. The DeKey method is introduced to reduce the key size. It distributes the convergent key across multiple servers using the Ramp Secret Scheme (RSS) instead of the key managed by the user. It splits the secret key into “n” shares and distributes it to multiple servers, such that any “k” shares can recover the secret key. It is difficult to manage all the server’s keys. If a key of a data block is shared among “n” servers, then the complexity of handling and managing the key at those servers is increased. Communication between the servers on handling user keys is also increased, which leads to increased communication overhead [25–27]. The summary of the literature survey is presented in Table 1.

In all the previous deduplication methods, a dedicated key server is used to generate and manage keys for cloud users. Qi et al. [28] have implemented an encrypted deduplication scheme for multiple key servers. Liu et al. [10] have introduced the idea of target deduplication by performing an attribute-keyword search on the ciphertexts. The results are quite promising, but the computation overhead of searchable encryption is very high compared to the normal attribute-based encryption method. Here, outsourcing decryption is used to optimize the scheme. This searchable encryption-based deduplication scheme is implemented only for text documents.

2.2.1. Homomorphic Deduplication. Muqel et al. [29] have implemented the homomorphic operations on the encrypted ciphertext to identify the redundant data blocks in the cloud storage. This homomorphic-based deduplication is to overcome the convergent deduplication technique problems. This method deployed a dedicated key server at the premises of the cloud service provider called as HEDup (homomorphic encryption deduplication). The cloud user encrypts the data with the keys provided by the HEDup key server. Here, internal adversarial attacks are possible where the keys are generated by the key-server present on the CSP. It also has large storage and latency overhead in maintaining the ciphertext.

Liu et al. [30] have introduced searching on encrypted data. The traditional encryption method will not allow the user or CSP to perform any kind of operation on the ciphertext. But homomorphic encryption technique allows the CSP to search, add, and multiply (somewhat homomorphic encryption/partial homomorphic encryption) the ciphertext. It uses searchable homomorphic encryption with tags and matching keywords used to perform deduplication. Youn et al. [31] used a challenge-response protocol and a third-party auditor to ensure the security of the entire system. To perform challenge-response protocols, a homomorphic linear authenticator is created based on the BLS signature [17].

3. Preliminaries: Homomorphic Encryption

Homomorphic encryption (HE) is a technique, where computational operations are carried by cloud service providers on top of the ciphertext without modifying the data format or compromising the security of the user data. A function \( f:G \longrightarrow H \) between two groups is homomorphic when \( fun(xy) = fun(x) \cdot fun(y) \forall x, y \in G \). Here, \( fun \) is a function, which takes the input from a group and performs an operation (addition and multiplication) to map with the other set.

Implementing homomorphic applications on cloud storage is a time-consuming process, but it ensures the security of the user data in the cloud environment. It also allows the cloud service provider to perform computations on the ciphertext. Rivest et al. (1976) and Rivest et al. (1978) implemented the first practical homomorphic encryption (RSA algorithm) in 1976 [32, 33]. But in the early 1980s, the computation power of the servers and systems was not capable of performing homomorphic encryption. The improvement in the computation power is achieved by homomorphic operations perform in cloud storage. In 2009, Gentry [34] has implemented fully homomorphic encryption on cloud storage based on ideal lattices. After the successful implementation of Craig Gentry’s (Stanford Ph.D. thesis 2009) work, homomorphic operations have become an important, futuristic technique in cloud computing. Some of the recent works on homomorphic encryptions are Cominetti et al. [35], Chou et al. [36], and Turan et al. [37].

3.1. Additive Homomorphic Encryption. An encryption scheme is called additive homomorphic encryption, if and only if, \( (a + b) = E(a) + E(b) \forall a, b \in n \) where \( E \) is encryption
and \( n \) is set of all possible messages. To develop a practical additive homomorphic encryption (PHE), additive or multiplicative functions are the only options to perform a homomorphic operation on top of the encrypted data because any Boolean circuit can be designed only through the XOR and NAND gate, where XOR performs the addition and NAND performs the multiplication. Examples of additive homomorphic encryptions are Pallier’s encryption [38] and Elgamal encryption [39] in which the plaintexts are encoded in the exponents.

### 4. Problem Statement

As discussed in the related works, the existing deduplication schemes have three major challenges, such as

(i) Performing source-level deduplication on the cloud storage server results in increased network bandwidth wastage

(ii) Due to the inability of scaling the index size, the false-positive errors are more in the source-level deduplication

(iii) The convergent key-based deduplication models have a high probability of information leakage and are vulnerable to confirmation of file attacks (CFA)

To overcome these issues, the proposed FogDedupe framework implements source-level and post-progress deduplication simultaneously. To reduce bandwidth wastage, the source-level deduplication is performed on the fog nodes that are kept closer to the data owners. Also, a Bloom filter-based distributed index table (DIT) is created and managed in the fog layer to perform source-level deduplication. It uses the master-slave protocol to frequently update the index table. In addition, a multi-key homomorphic encryption method-based scheme is proposed to perform post-progress deduplication which efficiently overcomes the vulnerabilities against CFA attacks.

### 5. FogDedupe Framework

The proposed FogDedupe framework performs both source-level and post-progress deduplication. The entities that are involved in the proposed deduplication scheme are (i) data owners, (ii) fog layers and fog nodes, and (iii) cloud service providers (CSP). Figure 1 describes the overall design and the entities of the proposed FogDedupe deduplication framework.

Data owners (DO) are the one who creates the data and uploads it to the cloud storage. The data owners are accountable and eligible to decide who can access the information stored in the cloud within their functional limits. To retrieve the data faster and to perform source-level deduplication, the data owner hashes the data blocks with one-way hash functions and sends the message digest to the nearby fog nodes [40, 41]. Also, the data owner creates public and private keys and encrypts the data blocks using the homomorphic encryption technique. Later, it sends the cipher text to

<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Technique used</th>
<th>Description</th>
<th>Findings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Halevi [13]</td>
<td>Verifying the proof of ownership (PoW) to overcome the security vulnerabilities</td>
<td>(i) Uses message digest to perform source-level deduplication and to verify the ownership of the data block</td>
<td>(i) Uses Merkle tree to verify the ownership (ii) Index table is not scalable</td>
</tr>
<tr>
<td>Blasco et al., [14]</td>
<td>Proof of ownership for deduplication using Bloom filter</td>
<td>(i) Cloud server maintains a Bloom filter to verify the ownership of the data by sending a challenge to the clients</td>
<td>(i) Traditional Bloom filter is not scalable (ii) The possibility of false-positive error is high</td>
</tr>
<tr>
<td>Agarwala et al. [17]</td>
<td>Client-side secure image deduplication using DICE protocol</td>
<td>(i) Data blocks of the images are encrypted using MLE and DICE protocol (ii) Redundant blocks in the image are removed</td>
<td>(i) Ensures security against poison attacks (ii) It fails to perform source-level deduplication when the same data block is received from different users</td>
</tr>
<tr>
<td>Youn et al. [18]</td>
<td>Authorized client-side deduplication using CP-ABE in cloud storage</td>
<td>(i) Provides authorized convergent encryption method using CP-ABE (ii) Allows only authorized users to access critical data</td>
<td>(i) A third-party server is introduced for generating convergent keys. It increases security vulnerabilities</td>
</tr>
<tr>
<td>Yoosuf et al. [19]</td>
<td>LDuAP: lightweight auditing with inline deduplication</td>
<td>(i) Two-dimensional Bloom filter is created and managed on the cloud servers to perform inline deduplication</td>
<td>(i) Bandwidth issues are not addressed as the inline deduplication is performed on the cloud servers</td>
</tr>
<tr>
<td>Bellari et al. [21]</td>
<td>DupLESS: server-ided encryption for deduplicated storage</td>
<td>(i) A separate key server is introduced to generate keys for the users</td>
<td>(i) If the authenticity of the key server is compromised, then the entire data present in the cloud may get leaked or misused</td>
</tr>
<tr>
<td>Chen et al. [24]</td>
<td>BL-MLE: block-level message-locked encryption for secure large file deduplication</td>
<td>(i) Reduces the metadata that is generated by the MLE method (ii) Allows to perform both file-level and block-level target deduplication</td>
<td>(i) Uses Bellaro et al.’s [21] method for POW</td>
</tr>
</tbody>
</table>
the cloud storage servers along with the non-redundant tag (NR) generated from source-level deduplication.

Fog layer is a cloud entity that acts as an intermediate layer between the DO and CSP. It consists of several fog nodes that are geographically dispersed and kept closer to the data owners. Also, a dynamically scalable distributed index table (DIT) is created and managed in the fog layer. Upon receiving a source-level deduplication request from the DO, the fog node verifies the index and creates tags. If the data block is unique, the fog node creates a nonredundant tag (NR) and sends it to the DO. If it is the redundant block, the fog node prohibits uploading the data block to the cloud storage server.

Cloud service provider (CSP) is the one who provides computing (hardware and software) services to cloud users. CSP has an unlimited resource capacity to store and process the uploaded data. It performs three major tasks:

(i) Verifies whether the ciphertext has authentic tags or not
(ii) Stores the ciphertext in the cloud storage server
(iii) Frequently performs target-level deduplication on the cloud storage servers [23]

5.1. Generating Partial Hash Values (pa) for Data Blocks. Initially, the data owner fragments the large size of data into several data chunks each with the size of 1024 KB. Later, for each data chunk, the data owner generates the partial hash pa. The data owner hashes the data chunks using n number of collision-resistant one-way hash functions (HF). As a result, each hash function generates a corresponding digest of size L bits. In the existing inline deduplication schemes, the data owner must share the entire L bit of message digests to the CSP to verify the non-redundancy in the index table. However, it increases the possibility of confirmation of file attacks. To overcome this issue, the proposed fog-centric inline deduplication scheme uses a partial hash value pa instead of the entire message digest. Algorithm 1 explains the process of generating partial hash values for the data chunks.

The data owner creates partial hash values for each data chunk and sends them to the nearby fog node. Sending redundant ciphertext directly to the cloud storage increases the communication overhead as well as the storage overhead. To reduce the wastage of bandwidth the proposed framework directs the data owners to send the computed partial hash values (pa) to the nearby fog node. Using the distributed index table, the fog node verifies the incoming data chunk and generates a tag for each data chunk. If a data block is non-redundant, then the fog node creates a non-redundant tag (NR) and sends it to the corresponding data owner. If the data block is redundant, a redundant tag (R) is sent to the data owner. Upon uploading the ciphertext of the data block, the tags have to be sent along with it. Algorithm 2 explains the source-level deduplication on the fog layer.

After receiving the partial hash values, the fog node calculates its corresponding hash bits and stores it in the distributed index table (DIT). The values in the DIT will
either be 0 or 1. Here, 0 represents that no data chunks have been accommodated in the particular location, and 1 represents that the location is already accommodated.

Consider that the data owner uses three hash functions and creates three partial hash values for a data chunk. If all the three corresponding bits of partial hash values (\(p_\alpha\)) is 1, then it is determined as a replicated data chunk. If any two of three (2/3) corresponding hash bits in the index table are 1 and one of three (1/3) hash bits are 0, then also it will be considered as a non-duplicated data chunk. Though it is considered a non-duplicated data chunk, there is a possibility for redundancy. To monitor these high-risk data chunks, the fog node calculates the percentage of corresponding non-zero hash bits in the index table and sends them to the cloud administrator very often.

5.2. Distributed Index Table. Managing a standard bloom filter in the fog nodes to perform source-level deduplication results in increased false-positive errors as it follows a one-dimensional data structure. Also, the standard bloom filters

---

**Algorithm 1: Data owner—generating partial hash values.**

**Input:** Data chunks DC \(_{1, 2, \ldots, m}\) (each with the size of 1024KB)

**Output:** Partial hash values (\(p_\alpha\))

**Begin:**

1. Data owner HASHES the data chunks (DC \(_{i=1, 2, \ldots, m}\)) using ‘n’ number of hash functions.
2. Data owners stores resultant message digests (hash values) of the data chunks in their local storage.
3. Generate a partial hash value
   **Begin**
   a. Divide the hash values into ‘p’ partitions (i.e.) (L bits of message digest / ‘p’ partitions).
   b. To derive the partial hash values ‘\(p_\alpha\)’, choose odd partitions from the message digest and take the even bits from each partition.
   c. Combines the even bits from each odd partition and create a partial hash value ‘\(p_\alpha\)’ for each data chunk.
   d. Creates chunk id (C\(_{id}\)), file id (F\(_{id}\)) for partial hash values.
   **End**
4. Transfer the chunk id (C\(_{id}\)), file id (F\(_{id}\)), and its corresponding partial hash values ‘\(p_\alpha\)’ to the nearby fog nodes.

---

**Algorithm 2: Fog node—identifying the redundant data chunks.**

**Input:** Partial hash values (\(p_\alpha\))

**Output:** Redundancy verification results.

**Begin**

1. Fog node receives the chunk id (C\(_{id}\)), file id (F\(_{id}\)), and hash values (\(p_\alpha\)) from the data owners.
2. Fog node identifies the location of each data chunk and stored it in the DIT.
3. Redundancy verification (incoming data chunks)
   If (all corresponding bit position of \(p_\alpha\) == 1)
   a. Data chunk ← Duplicate data chunk
   b. Create a tag to represent that the corresponding ciphertext of the data chunk is duplicate and the fog node prohibits the data owner from uploading redundant data to the cloud servers.
   c. Sends tag to the data owner.
   Else (bit position (SIT) == 0 or bit position (SIT) == at least one 0)
   a. Data chunk ← Non-duplicated data chunk
   b. Calculate the percentage of non-zero hash bits in the index table.
   c. Creates tag for the non-duplicated data chunk
   d. Sends the non-duplicated tag to the data owner.
   e. Sends the percentage of non-zero hash bits to the cloud admin.

---

**Input:** Data chunks DC \(_{1, 2, \ldots, m}\) (each with the size of 1024KB)

**Output:** Partial hash values (\(p_\alpha\))
do not support the scalability feature. However, the velocity of incoming data is very high in cloud services. So, managing a standard bloom filter in the fog nodes might result in a bottleneck situation.

To reduce the false-positive errors in identifying duplicate data chunks in the fog nodes, the proposed scheme creates a two-dimensional scalable index table that is distributed among all the fog nodes in the fog layer. When a request is received from the data owner to perform source-level deduplication, the fog node immediately accesses the distributed index table and sends the response as a tag to the DOs. Determining the initial size of the scalable index table in the fog node, the following formula is used:

\[
m = \sqrt{\frac{-\text{Total number of data chunks} \times \ln (\text{False positive rate})}{\ln (2)^2}}.
\]

The initial size of the index table is determined based on the data chunks received at a particular period. Later, based on the velocity of incoming data chunks, the size of the index table is increased.

\[
\text{Input: Growth rate, false-positive rate, and the number of hash functions used.}
\]

\[
\text{Output: Creating a new index table with increased size.}
\]

\[
\begin{aligned}
\text{Begin} & \\
1. \text{Find fill ratio of existing DIT, } n = m \times \ln 2^2 / \ln P & \\
2. \text{If (fill ratio > =50)} & \\
\text{Then} & \\
\text{If (growth rate == High)} & \\
\text{Then} & \\
& \text{Create a new index table with a size of 200% of } m \times n \\
& \text{Insert stream bits to the new index table} & \\
\text{Else} & \\
& \text{Create a new index table with a size of 100% of } m \times n \\
& \text{Insert stream bits to the index table} & \\
\text{Else} & \\
& \text{Return null and do not create a new index table} & \\
\text{End} &
\end{aligned}
\]

Algorithm 3: Fog node—scalability in the proposed distributed index table.

Figure 2: Updating the distributed index table.
5.3. Scalability of the Index Table. The proposed distributed index table (DIT) is capable of scaling its size from $m \times m$ to $2m \times m$ when the velocity of the data increases. The fog nodes continuously monitor the remaining available free slots in the index, i.e., the number of 0s in the index table is continuously monitored. If the non-accommodated slots in the index table go below a certain limit, then a large index table is generated on the same fog node. If the velocity of incoming data is less, then the newly generated index table is two times larger than the old index table and four times larger if the velocity is high. The false-positive rate of the newly created index table is always lesser than the old index table as it has a larger size. Algorithm 3 discusses the scalability of distributed index table in the fog nodes.

5.4. Updating Distributed Index Table Using Master-Slave Protocol. To perform source-level deduplication, a distributed index table is introduced in the FogDedupe architecture. Yoosuf et al. [42] suggested the source-level deduplication in the fog node. However, the index tables were managed in the fog nodes themselves, and it has a high risk of unavailability. To overcome this issue, the proposed FogDedupe framework introduces a distributed index table (DIT), where the same copy of the index table is present in all the fog nodes of the cluster. Using distributed index table (DIT) in the fog layers efficiently overcomes the unavailability issue. Figure 2 depicts the workflow of the proposed distributed index table, and Algorithm 4 explains the process of dynamically updating the records in the DIT.

5.5. Multi-Key Homomorphic Encryption-Based Target Deduplication. After receiving the non-redundant tag (NR) from fog nodes, the DO encrypts the data chunks using the additive homomorphic encryption method. Existing homomorphic encryption methods can perform computational operations on top of the ciphertext only if the data blocks are executed using the same public and private keys. It opens a path to information leakage and makes the ciphertext vulnerable to confirmation of file attacks (CFA). To overcome this issue, the proposed deduplication scheme allows the data owners to encrypt the data blocks using different ciphertexts.

In the key generation model, the data owner creates two vector keys $\overrightarrow{a}$ and $\overrightarrow{b}$ in which $\overrightarrow{a}$ is used as a private key and $\overrightarrow{b}$ is used as an offset value to perform an additive homomorphic operation by the cloud service provider. After creating the keys, the data owner uses a one-time pad encryption method to encrypt the data blocks. To perform an additive operation on the ciphertext “modulo n” operation is used. However, the computation overhead of executing homomorphic encryption-based deduplication remains high in the cloud environment. Continuously performing target deduplication in cloud storage servers is impractical and may lead to an extensive workload for the cloud service provider. So, in the proposed target deduplication method, the CSP calculates the high-risk data blocks, i.e., the data chunks that have a high possibility to be redundant are calculated, and only for those data chunks, the target deduplication is performed. The high-risk data blocks are identified from the information received from the fog nodes. Algorithm 5 explains the proposed multi-key homomorphic encryption algorithm to encrypt the non-redundant data block.

The proposed deduplication scheme uses an additive homomorphic encryption-based algorithm to perform deduplication. It effectively identifies the redundant block in the cloud storage servers by comparing the corresponding ciphertext. However, executing an additive homomorphic operation on the ciphertext stored in the cloud puts an extra workload on the CSP.

6. Performance Evaluation

The prime objectives of the proposed FogDedupe frameworks are (1) reducing false-positive errors in the index table
6.1. False-Positive Error Rate. The false-positive error in the proposed fog-centric inline deduplication refers to a situation in which an element points to a particular location as that of another element in the index table. The probability of false-positive error in the proposed index table is

\[
\text{Probability (false Positive)} = \left(1 - e^{\frac{k \cdot m}{n}}\right)^k, \quad (2)
\]

\[
\text{Expected (false positive)} = \sum_{i=0}^{n-1} Cf (i). \quad (3)
\]

Here, \(Cf\) represents the collision in the index table and \(i\) is the factor of collision. The false-positive rate of the fog-centric inline deduplication highly depends on the number of collisions that happens in the index table. Since the proposed index table is capable of scaling its size, the probability of hash collision is always lesser than the standard Bloom filter-based index tables. Table 2 depicts the false-positive error rate in the distributed hash table.

The probability of false-positive errors ranges between 0.002 and 0.004. It is very low compared to the standard bloom filter. Table 3 shows the probability of the false-positive errors after scaling the size of the index table.

The probability of false-positive error after scaling the size of the index table is always lesser than the initial size of the index table. So, the proposed fog-centric inline
deduplication can effectively be performed even if the velocity of the data increases.

6.2. Security Analysis of the Proposed Scheme. The existing deduplication algorithm uses the entire message digest of the data block to generate the encryption keys. It makes the existing schemes vulnerable to the confirmation of file attacks (CFA). To overcome this issue, instead of using the message digest of the data blocks to generate keys, the proposed scheme derives the partial hash values, i.e., \( L \) bits of message digest/\( p \) partitions from the message digest are sent to the fog nodes. Sharing the partial hash values to the fog node allows them to perform source deduplication using the scalable index table and tightens the security. Even if the partial hash values get leaked from the fog node, no intruder can match the remaining hash bits. Let us consider a data block of size 1024 KB is hashed and produces 256 hash bits. Here, instead of sending the entire 256 bits to the fog node, the proposed deduplication method derives partial hash values \( \text{pa} \) from 256 bits, i.e., 256 bits\( / \text{No. of p partitions} \) of bits are shared to the fog nodes. From these partial hash bits, it is impossible to perform confirmation of file attacks. Moreover, the proposed system uses \( k \) hash functions to derive the partial hash values. So it makes it more secured against hash collision attacks.

On the other hand, the proposed multi-key additive homomorphic encryption allows the CSP to execute computational functions on top of the ciphertext stored in the cloud storage servers. Though it increases the computational overhead, it also tightens the security of the data in the aspect of both internal and external attacks.

7. Implementation and Result Discussion

To assess the performance of the proposed FogDedupe framework, an open source Eucalyptus software is installed on an Intel Xeon E5 2620 server that has a processing speed of 2.1GHz with 64GB RAM. Eucalyptus-based private cloud setup consists of cloud controller (CLC), cluster controller (CC), and walrus (W). The cloud controller is responsible for performing administrative operations of the CSP. The cluster controller controls the cluster nodes connected to the main cloud server. Two personal computers with Intel i5 -7th gen processor and 8GB RAM are used to create the cluster nodes. The walrus represents the storage servers of the Eucalyptus private cloud. A total of 4 TB storage space with RAID 5 configuration is used as a storage server. The Eucalyptus open-source software is compatible with Amazon AWS and well-suitable for evaluating fog-based source-level deduplication. Also, the fog nodes are created between cloud storage servers and the DOs by installing FOG project v1.5.9 Intel i5 -7th gen processor with 8GB RAM.

The data chunking process and the generation of partial hash values for the data chunks are carried out by the data owner. Operations like data chunking, key generation, encryption, and creation of partial hash values were written in the python programming language. An open-source mhealth (mobile health) dataset from UCI repository is used to assess the proposed fog-centric deduplication scheme. It comprises 172,824 IoT healthcare sensor values.

The prime objective of the proposed work is to reduce the communication overheads and improve storage efficiency. To assess the proposed scheme, the communication overhead and computation overhead on the fog nodes to perform inline deduplication and the computation overhead on the CSP to execute additive homomorphic operations on the ciphertext stored in the cloud are measured. Also, the redundancy elimination ratio of the proposed scheme is compared with BL-MLE, DupLESS, Youn et al. [18] deduplication schemes.

7.1. Communication Overhead. Two different scenarios are considered to measure the communication overhead for performing inline deduplication. (a) The data owner uploads the ciphertexts of the data chunks directly to the cloud, i.e., no fog layers are formed to perform inline deduplication. (b) The data owner verifies the redundancy of the data chunks in fog nodes first and then sends the ciphertext to the cloud.

Introducing a fog layer between DOs, the cloud effectively reduces the communication delay to a maximum of 60%, because the proposed FogDedupe source-level deduplication framework prohibits DOs to upload redundant ciphertext to the cloud storage servers directly.

The fog nodes are usually kept near to the DOs to quickly perform source-level deduplication, and the ciphertext of the DOs data is stored in the Amazon AWS-walrus storage. Table 4 shows the communication time required to access the fog node and the walrus storage. Results show that introducing fog nodes in the cloud services reduces the communication overhead by a maximum of 60%.

The communication overhead of the data owner transferring the \( \text{pa} \) to the fog nodes and uploading the ciphertext to the cloud storage is measured and shown in Figure 3.

7.2. Computation Overhead on the Fog Nodes. Fog nodes receive the hash value \( \text{pa} \) from the data owners and compute

<table>
<thead>
<tr>
<th>Table 2: False-positive errors in the distributed index table.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of data blocks</strong></td>
</tr>
<tr>
<td>---------------------------</td>
</tr>
<tr>
<td>1024</td>
</tr>
<tr>
<td>2048</td>
</tr>
<tr>
<td>3072</td>
</tr>
<tr>
<td>4096</td>
</tr>
<tr>
<td>5120</td>
</tr>
<tr>
<td>10240</td>
</tr>
<tr>
<td>102400</td>
</tr>
<tr>
<td>2048000</td>
</tr>
<tr>
<td>5120000</td>
</tr>
<tr>
<td>10240000</td>
</tr>
<tr>
<td>20480000</td>
</tr>
</tbody>
</table>
### Table 3: Probability of false-positive error after scaling the size of index table.

<table>
<thead>
<tr>
<th>Data blocks</th>
<th>Total bits in index table (before scaling) ( (m_{\text{prev}}) )</th>
<th>Prob. of false-positive before scaling its size ( (f_{P_{\text{prev}}}) )</th>
<th>Total bits in the index table after scaling its size ( (m_{\text{new}}) )</th>
<th>Min (GR)</th>
<th>Max (GR)</th>
<th>Prob. of false-positive ( f_{P_{\text{new}}}(\text{new matrix}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>9801</td>
<td>0.003319</td>
<td>19602</td>
<td>39204</td>
<td>2.6 ( \times 10^{-3} )</td>
<td>2.9 ( \times 10^{-3} )</td>
</tr>
<tr>
<td>2048</td>
<td>19600</td>
<td>0.003319</td>
<td>39200</td>
<td>78400</td>
<td>2.6 ( \times 10^{-3} )</td>
<td>2.9 ( \times 10^{-3} )</td>
</tr>
<tr>
<td>3072</td>
<td>29241</td>
<td>0.002969</td>
<td>58482</td>
<td>116964</td>
<td>2.3 ( \times 10^{-3} )</td>
<td>2.6 ( \times 10^{-3} )</td>
</tr>
<tr>
<td>4096</td>
<td>39204</td>
<td>0.003319</td>
<td>78408</td>
<td>156816</td>
<td>2.6 ( \times 10^{-3} )</td>
<td>2.9 ( \times 10^{-3} )</td>
</tr>
<tr>
<td>5120</td>
<td>48841</td>
<td>0.003058</td>
<td>97682</td>
<td>195364</td>
<td>2.4 ( \times 10^{-3} )</td>
<td>2.7 ( \times 10^{-3} )</td>
</tr>
<tr>
<td>10240</td>
<td>97969</td>
<td>0.00319</td>
<td>195938</td>
<td>391876</td>
<td>2.5 ( \times 10^{-3} )</td>
<td>2.8 ( \times 10^{-3} )</td>
</tr>
<tr>
<td>102400</td>
<td>980100</td>
<td>0.00321</td>
<td>1960200</td>
<td>3920400</td>
<td>2.5 ( \times 10^{-3} )</td>
<td>2.8 ( \times 10^{-3} )</td>
</tr>
<tr>
<td>204800</td>
<td>1960000</td>
<td>0.00325</td>
<td>3920000</td>
<td>7840000</td>
<td>2.6 ( \times 10^{-3} )</td>
<td>2.9 ( \times 10^{-3} )</td>
</tr>
<tr>
<td>512000</td>
<td>4906225</td>
<td>0.00312</td>
<td>9812450</td>
<td>19624900</td>
<td>2.5 ( \times 10^{-3} )</td>
<td>2.8 ( \times 10^{-3} )</td>
</tr>
<tr>
<td>1024000</td>
<td>9809424</td>
<td>0.00317</td>
<td>19618848</td>
<td>39237693</td>
<td>2.5 ( \times 10^{-3} )</td>
<td>2.8 ( \times 10^{-3} )</td>
</tr>
<tr>
<td>2048000</td>
<td>19624900</td>
<td>0.00319</td>
<td>39249800</td>
<td>78499600</td>
<td>2.5 ( \times 10^{-3} )</td>
<td>2.8 ( \times 10^{-3} )</td>
</tr>
</tbody>
</table>

### Table 4: Communication time required to access AWS storage.

<table>
<thead>
<tr>
<th>Number of Data blocks</th>
<th>Data owner directly accessing AWS storage (without fog node)</th>
<th>Data owners accessing the AWS with fog nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time in milliseconds</td>
<td>Standard deviation in milliseconds</td>
</tr>
<tr>
<td>5</td>
<td>320.26</td>
<td>136.26</td>
</tr>
<tr>
<td>25</td>
<td>1350.76</td>
<td>540.45</td>
</tr>
<tr>
<td>50</td>
<td>2750.50</td>
<td>994.21</td>
</tr>
<tr>
<td>100</td>
<td>5556.93</td>
<td>1899.01</td>
</tr>
<tr>
<td>200</td>
<td>11789.12</td>
<td>3444.56</td>
</tr>
</tbody>
</table>

![Figure 3: Communication overhead.](image-url)
Figure 4: Comparison of computation overhead on fog node with existing additional key server schemes.

Figure 5: Storage efficiency of cloud storage servers.

Figure 6: Comparison of deduplication ratio.
its corresponding locations in the distributed index table (DIT). In the proposed deduplication scheme, fog nodes do not generate any keys on behalf of the data owner; rather, it simply calculates the bit position from the partial hash values and changes the values from 0 to 1. However, the existing deduplication schemes use a separate key server on the third parties to create convergent keys to encrypt the chunks.

The computation overhead of the proposed deduplication scheme in the fog node is much lesser than the existing methods. Figure 4 compares the computation time required in the fog nodes to execute the proposed inline deduplication with existing key server deduplication schemes.

To evaluate the proposed scheme, 25,000 IoT healthcare IoT sensor values are randomly chosen. The data owner hashes each sensor value and creates $p_x$. Later, they are communicated to the nearby fog node. The $p_x$ is stored in the DIT, and the fog node dynamically performs source-level deduplication. The deduplication ratio of the proposed source-level deduplication is calculated using the following formula:

$$\text{Deduplication Ratio} = \frac{\text{Size of the input data} - \text{Storage utilized}}{\text{size of the input data}}.$$ \hfill (4)

To assess the performance of the proposed scheme, two different scenarios are considered. (i) The randomly selected 25,000 data chunks from the mhealth dataset are encrypted, and the ciphertext is uploaded to the walrus directly by the DO, i.e., no deduplication algorithm is executed. (ii) The randomly selected 25,000 sensor values are uploaded to the cloud storage after performing fog-centric source level deduplication.

The storage utilization of the cloud storage server is calculated for both scenarios. After implementing the fog-centric source-level deduplication, the storage efficiency is increased up to 30%. Figure 5 shows the improvements in the storage efficiency after implementing the deduplication scheme.

The redundancy elimination ratio is calculated using the following formula:

$$\text{Redundancy Elimination Ratio} = \frac{\text{Total number of correctly removed redundant sensor values}}{\text{Total number of Sensor values sent to fog nodes}}.$$ \hfill (5)

Figure 6, compares the ratio of duplicate data that are removed by BL-MLE, DupLESS, Zhen et al. 2017, Youn et al. 2019, and the proposed fog-centric inline deduplication. The redundancy elimination ratio of the proposed deduplication scheme ranges from 0.84 to 0.93. It reduces the storage overhead of the cloud by approximately 30% by maintaining the redundancy elimination ratio between 0.85 and 0.94.

8. Conclusion and Future Works

To reduce the network bandwidth wastage and to overcome the false-positive errors in the source-level deduplication, a FogDedupe framework is proposed. It performs both source-level deduplication and post-progress deduplication to improve the efficiency of the cloud service and storage servers. To perform source-level deduplication a fog layer, consisting of "n" number of fog nodes is introduced. Also, a distributed index table is created and managed on the fog layer. The fog nodes present on the same cluster use master-slave protocol to update the index values. The scalability feature in the proposed distributed index table effectively reduces the false positive errors in the source-level deduplication. Likewise, to perform target-level deduplication, a multi-key additive homomorphic method is introduced. Though the post-progress deduplication takes a slightly larger time to identify the duplicate blocks, it overcomes the security threats raised by the CFA attacks effectively. In the future, instead of using additive homomorphic operations, fully homomorphic operations are planned to be implemented on the cloud storage server to perform post-progress deduplication.

Data Availability

The data that support the findings of this study are available from the corresponding author, upon reasonable request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References


[38] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in In International conference on


