Hindawi
Security and Communication Networks
Volume 2018, Article ID 2193897, 16 pages
https://doi.org/10.1155/2018/2193897

Research Article

A Symmetric Key Based Deduplicatable Proof of Storage for Encrypted Data in Cloud Storage Environments

Cheolhee Park,[1] **Hyunil Kim**,[2] **Dowon Hong**,[1] **and Changho Seo**[1,2]

[1]*Department of Mathematics, Kongju National University, 32588, Republic of Korea*
[2]*Department of Convergence Science, Kongju National University, 32588, Republic of Korea*

Correspondence should be addressed to Dowon Hong; dwhong@kongju.ac.kr

Received 7 June 2018; Revised 18 September 2018; Accepted 15 October 2018; Published 1 November 2018

Academic Editor: Zhe Liu

Over the recent years, cloud storage services have become increasingly popular, where users can outsource data and access the outsourced data anywhere, anytime. Accordingly, the data in the cloud is growing explosively. Among the outsourced data, most of them are duplicated. Cloud storage service providers can save huge amounts of resources via client-side deduplication. On the other hand, for safe outsourcing, clients who use the cloud storage service desire data integrity and confidentiality of the outsourced data. However, ensuring confidentiality and integrity in the cloud storage environment can be difficult. Recently, in order to achieve integrity with deduplication, the notion of deduplicatable proof of storage has emerged, and various schemes have been proposed. However, previous schemes are still inefficient and insecure. In this paper, we propose a symmetric key based deduplicatable proof of storage scheme, which ensures confidentiality with dictionary attack resilience and supports integrity auditing based on symmetric key cryptography. In our proposal, we introduce a bit-level challenge in a deduplicatable proof of storage protocol to minimize data access. In addition, we prove the security of our proposal in the random oracle model with information theory. Implementation results show that our scheme has the best performance.

1. Introduction

Cloud storage is an attractive service where clients can outsource data to a remote storage and access the outsourced data anywhere, anytime. Moreover, clients can reduce the large burden of local storage via cloud storage. Due to these advantages, cloud storage services are becoming increasingly popular. Therefore, the data stored in the cloud is explosively growing. According to a report, the volume of outsourced data in cloud storage is expected to reach 40 trillion GB in 2020 [1]. Among the outsourced data, most of them are duplicated [2]. Hence, cloud storage service providers can save huge amounts of storage space via deduplication, in which the cloud server maintains only a single copy of the redundant data and assigns a link of the data to all clients that own the same data. Moreover, through client-side cross-user deduplication, cloud servers can save not only storage space but also network bandwidth whereby the client directly checks data duplications to determine whether to transmit data. Therefore, cloud servers can save storage space and network bandwidth.

Despite these advantages, general client-side deduplication is vulnerable to a number of threats, as only a short and fixed identifier (e.g., hash value of the data) replaces the whole data. For example, cloud servers can be used as a content distribution network unintentionally, and malicious clients can launch a poison attack (Target Collison attack), etc. [3, 4]. Thus, cloud servers that provide the client-side cross-user deduplication have to confirm whether the client actually owns the upload requested data [4].

In addition, through various incidents such as cloud data leakages and corruption, the security of outsourced data has become an important issue. Since the data in clouds can be attacked by internal and external attackers, personal privacy or secret information of enterprises can be leaked or corrupted, which can be fatal. Therefore, the cloud server has to ensure confidentiality and integrity of the outsourced data.

However, this can destroy the deduplication goal of using resources efficiently.

Firstly, in terms of confidentiality, if the client encrypts the data using conventional encryption systems, deduplication will fail. Since conventional encryption systems encrypt the data using different encryption keys for each user, it outputs different results even though the inputs are the same. To overcome this, convergent encryption was proposed, where the hash value of the data is used the encryption key [5]. With this, the same data becomes the same ciphertext after encryption, and it enables the cloud server to deduplicate while ensuring confidentiality. However, convergent encryption is vulnerable to brute-force attack. That is, convergent encryption can ensure confidentiality for only unpredictable data. Note that, without loss of generality, we assume that that the original data has enough entropy against message guessing attack as with [6].

Secondly, in terms of integrity, the cloud server does not intentionally damage the client's data, however outsourced data can be corrupted during the internal process of the cloud by unintentional error. The cloud server may hide the incident to the client to maintain their reputation. Hence, the client can require an audit for the integrity of outsourced data periodically. However, applying conventional integrity check techniques to the cloud system, such as message authentication codes, can create a huge burden to both the client and cloud server as it requires the local data to verify integrity. To overcome this problem, Ateniese *et al.* [7] introduced "provable data possession (PDP)" and Juels and Kaliski [8] introduced "proof of retrievability (PoR)", which enable remote data auditing probabilistically. However, in the perspective of deduplication, each client that has the same data must generate metadata in order to audit integrity. The cloud server then has to store all of their metadata. This can lead to a huge overhead of storage and can destroy the fundamental goal of efficient usage of resources.

Recently, in order to achieve the goal of deduplication while ensuring integrity in cloud storage environments, various techniques have been proposed [1, 2, 9–12]. However, [1, 2, 9] have heavy computational costs because the schemes are based on public key cryptography, and the client's privacy can potentially be leaked as they support publicly verifiable integrity auditing. This means that, with respect to a file, subsequent clients have to get the first uploader's public key to be used in integrity auditing. Therefore, every subsequent client can know who has the file. In the case of [10–12], new techniques that are not based on public key cryptography have been proposed, which are based on homomorphic operations. Compared to previous schemes based on public key cryptography, they are more efficient in terms of computation. However, they are still inefficient, and with large variations in efficiency depending on block and file size. Moreover, if the file size is small, almost all of the entire file needs to be checked (e.g., if the file size is less than 2 MB and block size is 4 KB, more than 89% of the entire file needs to be accessed). By capturing the issues mentioned above, we apply a bit-level challenge in order to achieve data access efficiency, even in the small data.

In this paper, we propose a secure and highly efficient deduplicatable proof of storage scheme based on symmetric key cryptography, namely Sec-DPoS, which ensures data confidentiality with brute-force attack resilience and supports integrity auditing based on symmetric key cryptography. In terms of secure client-side cross-user deduplication, we achieve a proof of ownership protocol by changing the role of prover and verifier in an integrity auditing protocol. In addition, we apply a bit-level challenge in an ownership check and integrity auditing protocol in order to support various file sizes with efficiency. Moreover, we prove the confidence of the detection probability for the bit-level challenge by information theory. We summarize the properties of our construction as follows:

(1) **Data confidentiality with dictionary attack resilience**. In terms of encrypted data deduplication, we exploit key server to ensure dictionary attack resilience, as with [1, 13]. The clients encrypt data using the encryption key distributed from the key server and the encryption key is derived from the data.

(2) **Integrity auditing with deduplication based on symmetric key cryptography**. To audit the integrity of the outsourced data, the client precomputes an expected response to be used later over encrypted data and uploads the expected responses to the cloud server in encrypted form. Our integrity auditing protocol is a motivated symmetric key based integrity auditing scheme. We apply a bit-level challenge to the symmetric key based integrity auditing scheme. In addition, since the expected response is generated using a message-derived key, a client with the same data can audit the integrity of the outsourced data without generating additional metadata.

(3) **Secure proof of ownership over encrypted data**. Proof of ownership is similar to the integrity auditing protocol but with the role of prover and verifier changed. In our construction, only clients with intact data can pass the proof of ownership protocol. Also, since the proof of ownership protocol is performed over encrypted data, the protocol does not expose any information of plaintext.

(4) **Privacy leakage resilience**. In the public key based solution, subsequent clients need to know the public key generated by the first uploader to audit the integrity. This means that subsequent clients can immediately learn who has a file. In our construction, we can prevent privacy leakage in the integrity auditing process as the scheme is based on symmetric key cryptography.

Our Contributions. The main contributions of our proposal can be summarized as follows:

(1) Sec-DPoS is the first approach of deduplicatable proof of storage based on symmetric key cryptography and is a secure and highly efficient deduplicatable proof of storage scheme with ensuring confidentiality. We analyze the security of Sec-DPoS and prove that our scheme is secure in the random oracle model.

(2) In contrast to the previous solutions, in order to ensure efficiency, even if the data is small, we adopt a bit-level challenge and prove confidence of detection probability in information theory.

(3) We implement and evaluate our proposal. The implementation results show that our scheme has the highest performance compared to other schemes.

The remainder of this paper is organized as follows: In Section 2, we briefly review related work. In Section 3, we present the system construction, security model and design goal. We then propose Sec-DPoS in Section 4 and analyze the security of Sec-DPoS in Section 5. In Section 6, we present the implementation of our scheme and compare with other schemes. Finally, in Section 7, we conclude the paper.

2. Related Works

In this section, we discuss previous work related to secure deduplication, integrity auditing and recent solutions that achieve the goal of both deduplication and integrity auditing.

2.1. Deduplication. In cloud storage environments, cloud servers can save storage space via deduplication, where the cloud server keeps only a single copy of the files. In addition, the server can save network bandwidth, as well as storage space, via client-side deduplication. However, general client-side deduplication has vulnerabilities as only a short and fixed identifier (e.g., hash value of a file) replaces the entire file [3]. Hence the server should verify that clients who intend to upload files have the intact file. For secure client-side deduplication, Halevi *et al.* [4] introduced the proof of ownership (PoW) notion, in which only the client who has an intact file can pass the ownership verification. They also proposed several PoW schemes whereby the client can efficiently prove ownership to the server based on the Merkle hash tree. Pietro and Sorniotti [14] proposed a more efficient PoW scheme, known as s-PoW. Since s-PoW needs only randomly chosen k position bits, the complexity of the protocol is independent from file size.

On the other hand, in order to achieve deduplication over encrypted data, Douceur *et al.* [14] proposed convergent encryption (CE) that uses the hash value of a file as an encryption key. Therefore, the same files create the same results after encryption. Similarly, Bellare *et al.* [6] proposed a cryptographic primitive known as message-locked-encryption (MLE). However, since both CE and MLE use the hash value of the file as an encryption key, they are vulnerable to dictionary attack (i.e., predictable data can be leaked by brute force attack). To overcome this problem, Keelveedhi *et al.* [13] proposed DupLESS in which clients encrypt data using a message-derived key via interaction with a key server. Since the encryption key is generated by an oblivious pseudorandom function and the key generation request is bounded by rate-limiting, outsourced data can be protected from brute force attack. Liu *et al.* [15] proposed an encrypted data deduplication scheme without additional independent servers, while ensuring brute force attack resilience. In [15], since the protocol is based on Password Authenticated Key Exchange (PAKE), clients who have the same file can share an encryption key without additional servers. In integrated network environments, Qi *et al.* [16] proposed an encrypted data deduplication scheme that improves the security and network latency by introducing many key servers in the network. In terms of combining the both functionalities, various studies have been conducted to satisfy data confidentiality and to support ownership check [17–20].

2.2. Integrity Auditing. From the perspective of the client, integrity auditing of the outsourced data is one of the important issues for secure outsourcing, as the outsourced data can be corrupted by unintentional errors. Ateniese *et al.* [7] proposed a notion of provable data possession (PDP) for ensuring integrity of remote data, in which the client can audit the integrity of the target file without maintaining the entire file. Ateniese *et al.* [21] proposed a highly efficient PDP scheme based on symmetric key cryptography, with the support of a dynamic scenario except insertion. To support dynamic scenario with insertion, Erway *et al.* [22] proposed dynamic-PDP based on a rank-based skip list. Wang *et al.* [23] proposed a proxy-PDP, in order to relax the computational overhead for tag generation. Zhu *et al.* [24] proposed a cooperative-PDP scheme in a multicloud environment. Based on convergence encryption, Liu *et al.* [25] proposed integrity auditing scheme and considered integrity tag deduplication over encrypted data.

In another way to support integrity auditing, Juel and Kaliski [8] proposed a notion of proof of retrievability (PoR), in which integrity auditing is performed using a sentinel inserted into the file. Compared with PDP, PoR support retrievability, as well as integrity auditing, yet it has a limitation in the number of queries. In order to achieve both private and public verifiability, Shacham and Waters [26] proposed two types of PoR schemes using a homomorphic authenticator. Wang *et al.* [27] proposed an improved PoR scheme based on the Merkle hash tree to achieve the goal of PoR in dynamic scenarios. Xu and Chang [28] proposed an improved PoR scheme to reduce communication costs. Aimed at specific conditions, Li *et al.* [29] proposed OPoR to support PoR over resource limited devices, and Ren *et al.* [30] proposed a PoR scheme in dynamic scenarios for coded cloud systems.

2.3. Secure Client-Side Deduplication with Integrity Auditing. As a method that provides both secure deduplication and integrity auditing, Zheng and Xu [9] firstly proposed proof of storage with deduplication (POSD), based on public key cryptography. However, an error in security occurs if the first uploader maliciously generates a pair of public and private keys [31], and POSD does not ensure confidentiality of the outsourced data as it is run over plaintext. Yuan and Yu [2] proposed a scheme called PCAD that supports both deduplication and integrity auditing with batch auditing, in which the server can simultaneously prove the possession of multiple files. Li *et al.* [1] proposed two schemes, namely SecCloud and SecCloud+. In both schemes, the author introduced an auditing entity that maintains a MapReduce cloud, which helps the client to generate block tags for

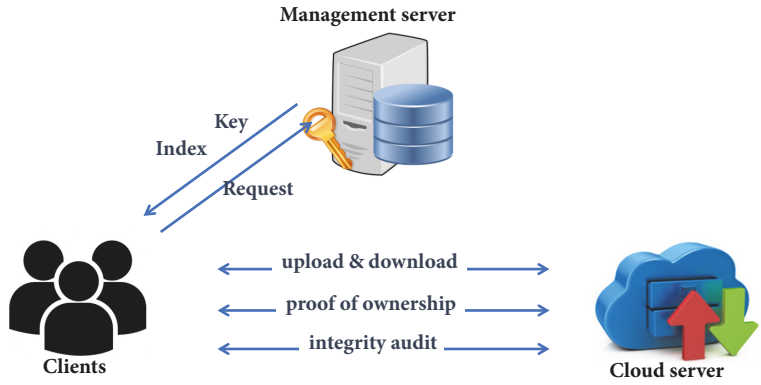

Figure 1: The Sec-DPoS system model.

integrity auditing. Additionally, SecCloud+ ensures confidentiality, where the client encrypts files using a message-derived encryption key distributed from the key server. In terms of efficiency improvement, Youn *el al.* [32] proposed a new scheme based on the homomorphic linear authenticator [26].

However, since the schemes in [1, 2, 9, 32] are based on public key cryptography, they have heavy computational cost and privacy can be leaked. For example, for the systems that support deduplication with public key based integrity auditing, subsequent uploaders must use the public key of the first uploader. In this case, the subsequent uploader can know who has the file, leading to privacy leakages.

As another approach, Du *et al.* [10] proposed proof of ownership and retrievability (PoOR) based on the Merkle hash tree and homomorphic verifiable tags. Compared with public key based schemes, PoOR is more efficient in terms of computational costs. As an improved PoOR, Chen *et al.* [11] proposed a Message-locked PoOR scheme that applies a message-derived key and supports remote repairing. Since clients who have the same file can generate the same convergent key, privacy leakage can be preserved. However, Message-locked PoOR causes unnecessary block access in the ownership check protocol as this is based on HMAC. He *et al.* [12] proposed a deduplicatable dynamic proof of storage scheme based on a homomorphic authenticated tree in order to support dynamic scenarios. However, the schemes in [11, 12] are still inefficient and vulnerable to dictionary attack. Moreover, there is a large variation in efficiency, depending on block and file size. If the file size is small, almost all of the entire file needs to be checked.

3. Models and Goal

In this section, we describe components and design the goal of our proposal. We first illustrate the system model and present a threat model that can occur in the cloud storage environment. In the description of the design goal, we present a trivial solution that is a simple combination of the secure client-side deduplication and symmetric key based integrity auditing scheme. Following this, by capturing problems of a trivial solution, we describe how our approach achieves deduplicatable proof of storage.

3.1. System Model. A structure of our proposed scheme consists of three entities as shown in Figure 1.

(i) The cloud server (*Srv*) provides cloud storage services. Typically, the cloud server operates a large storage space and computational resources. The cloud server attempts to minimize the bandwidth and to optimize the use of storage space via client-side cross-user deduplication. We assume that the cloud server is honest-but-curious.

(ii) The client (*Clt*) uses the cloud storage service provided from the cloud server. The client uploads data and has access to the outsourced data at all times.

(iii) The management server (*Msrv*) helps clients upload data. The management server distributes a message-derived secret key and manages the challenge index. We assume that the management server is a trusted third party (the functionality of the management server is described in detail at Section 4).

In our system, when a client wants to upload a file F, the client firstly interacts with the management server to get a secret key and challenge index. The client then generates an identifier of the file and sends an upload request with the identifier to the cloud server. The cloud server checks whether the file exists in storage. If the file exists in the cloud, the client does not need to upload the file and the cloud server provides a link of the file to the client after identifying whether the client actually has the file. If the file does not exist in the cloud, the client uploads the encrypted file to the cloud with the preprocessed information. After the upload, the client can audit the integrity of the outsourced data at any time.

3.2. Threat Model. In this subsection, we discuss various threats for the cross-user client-side deduplication environment within the remote data auditing system. In our system model, we assume that the cloud server is honest-but-curious. This means that the cloud server honestly performs system protocols yet it can curiously intrude client's privacy as it has access to the client's data. Moreover, the cloud server can be a victim of an outside attack. Hence, the client's data in the cloud server can potentially be leaked inside and outside. Thus, we design our scheme to ensure data confidentiality

with brute-force attack resilience by introducing the management server.

The cloud server does not intentionally damage outsourced data. However, the data in the cloud can be corrupted by unintentional system errors. When the data in the cloud is corrupted, the cloud server can hide the data loss incident to the client in order to maintain their reputation. Thus, we design our scheme to ensure that if the cloud server loses a part of the outsourced data, it cannot forge integrity to the client in the audit. Briefly, a cloud server that loses a part of the outsourced data cannot pass the integrity auditing protocol with a given probability (e.g., 99%). In Section 5, we prove the security of unforgeability in detail.

In the perspective of cross-user client-side deduplication, as mentioned previously, a malicious client that has only partial information of a file can claim possession in order to maliciously obtain the file. The malicious client can attempt to convince the cloud server of its ownership with the check protocol without the entire file. Thus, we design our scheme to ensure that malicious clients cannot cheat the cloud server in ownership checking. Briefly, a malicious client cannot pass the ownership checking protocol, except with a negligible probability. In Section 5, we prove the security of uncheatability in detail.

3.3. Design Goal. To achieve both integrity auditing and secure deduplication in practice, we considered Ateniese *et al.*'s scheme [21], a highly efficient integrity auditing scheme based on symmetric key cryptography. As a trivial solution, there is a simple method that combine Ateniese *et al.*'s scheme with the cross-user client-side deduplication system, as in the following case:

Trivial Solution. When clients intend to upload a file as a first uploader, they first generate, then sequentially arrange, expected responses for integrity auditing. The client then uploads the file and the arranged set of expected responses to the cloud. Note that the expected responses are encrypted using authenticated encryption before upload using a randomly chosen secret key. In this case, since the file is uploaded first, the cloud server generates metadata (expected responses used in the ownership check protocol) for secure deduplication and sends the file to the secondary storage.

Unfortunately, there are two major limitations. The first problem is that subsequent clients that have the same file cannot use the expected responses generated by the first uploader as it is encrypted by the first uploader's private key. Hence, subsequent clients have to generate another set of expected responses, which can lead to intense overheads in terms of storage space, network bandwidth and computational costs. The second problem is the management of the challenge index. Even if the first problem is resolved, there may be a collision problem of the challenge index. Under the assumption that the first problem is solved, if subsequent clients have the same file, they can use the arranged set of expected responses generated by the first uploader. However, every client who has the ownership of the file cannot know that what values are used. This means that certain expected responses can be used repeatedly. The cloud server can then launch a replay attack i.e., the cloud server can simply avoid the integrity auditing protocol by storing the pairs of used challenges and responses.

In order to overcome above problems, we exploit a management server, in which the client can get a message-derived key from the management server, as in [1, 13]. The key is used in the file encryption and integrity auditing. Moreover, we design the management server to handle the challenge index in order to avoid the challenge index collision problem.

With respect to ownership checking, we design our proof of ownership scheme to change the role of prover and verifier in the integrity auditing protocol. Thus, the cloud server generates expected responses for proof of ownership before transmission of the file to the secondary storage and retains the expected responses in the local storage.

As illustrated above, we achieve both the goal of secure client-side cross-user deduplication and integrity auditing based on symmetric key cryptography. In our construction, the cloud server can save both storage space and network bandwidth while efficiently ensuring data integrity and confidentiality.

4. Sec-DPoS: A Symmetric Key Based Deduplicatable Proof of Storage

In this section, we describe our proposal in detail. First, we illustrate about preliminaries and notations. Then, our proposed scheme is described in a detailed way. The components of our scheme consists of four protocols.

4.1. Preliminaries. Firstly, we describe the building blocks as follows:

(i) **Collision-resistant hash function**. A hash function $H : \{0,1\}^* \longrightarrow \{0,1\}^*$ is collision-resistant if it is impossible to find two different values x and y that satisfy $H(x) = H(y)$ and takes a binary string of arbitrary length as input, and outputs a binary string of fixed length.

(ii) **Key derivation function**. A key derivation function $KDF : \{0,1\}^* \times \{0,1\}^* \longrightarrow \{0,1\}^*$ is a deterministic function that takes a secret seed s and an input x and outputs a secret key.

(iii) **Pseudorandom function**. A pseudorandom function $f : \{0,1\}^* \times \{0,1\}^* \longrightarrow \{0,1\}^*$ is a deterministic function that takes a key k and an input x and outputs a value y that is indistinguishable from a truly random function of the same input x within the same range. We define $f_k(x) \stackrel{\text{def}}{=} f(k, x)$.

(iv) **Pseudorandom permutation**. A pseudorandom permutation $\pi_{key}(x) : \{0,1\}^* \times [0, M] \longrightarrow [0, M]$ is a deterministic function that takes a key k and an integer x where $1 \leq x \leq M$ and outputs an integer y where $1 \leq y \leq M$. It is indistinguishable from a truly random permutation of the same input x. In our construction, we use π to extract bit indices of a file F. Therefore, $M = |F|$ where $|F|$ denotes the bit length of the file F. We define $\pi_{key}(x) \stackrel{\text{def}}{=} \pi(k, x)$.

Input : a hash value h_F of a file F, a master key MK_{Msrv} of management server, a data table T of management server
Output : a secret key K_{h_F}, the highest challenge τ_{h_F}, an index subset I_α

(1) if $h_F \notin T$ then
(2) $K_{h_F} \longleftarrow H(MK_{Msrv} \parallel h_F)$
(3) $\tau_{h_F} \longleftarrow 0$
(4) $I^{h_F} \longleftarrow I$
(5) save ($key = h_F$; $value = K_{h_F}, \tau_{h_F}, I^{h_F}$) in T
(6) else
(7) load $value$ corresponding to $key = h_F$ from T
(8) if $I^{h_F} = \emptyset$ then
(9) $I^{h_F} \longleftarrow I$
(10) $\tau_{h_F} \longleftarrow \tau_{h_F} + t$
(11) randomly choose $I_\alpha \in_R I^{h_F}$
(12) return $(K_{h_F}, \tau_{h_F}, I_\alpha)$

Algorithm 1: The key and index distribution algorithm.

(v) **Deterministic symmetric encryption**. A deterministic symmetric encryption Enc takes a key k and a plaintext m as input and outputs a ciphertext. We use the notation $Enc_k(m)$ to denote the deterministic symmetric encryption algorithm.

(vi) **Authenticated encryption**. An authenticated encryption algorithm AE takes a key k and a plaintext m as input and outputs a ciphertext and authentication tag. We use the notation $AE_k(m)$ to denote the authenticated encryption algorithm.

Secondly, we describe our notation as follows:

(i) **Expected response set.** An expected response set $ResArr$ is an array that is a set of precomputed responses to audit the integrity of remote data. In particular, $ResArr_{Clt}$ is generated from the client to audit integrity of outsourced data and $ResArr_{Srv}$ is generated from the cloud server for proof of ownership, where $|ResArr_{Clt}| = t$ and $|ResArr_{Srv}| = s$.

(ii) **Challenge index.** A challenge index Idx indicates a specific location of $ResArr$. In particular, Idx_{clt} indicates a specific location of $ResArr_{clt}$ and Idx_{srv} indicates a specific location of $ResArr_{srv}$, where $1 \leq Idx_{clt} \leq t$ and $1 \leq Idx_{srv} \leq s$. Additionally, the client retains Idx_{clt} for integrity auditing and the cloud server retains Idx_{srv} for proof of ownership.

(iii) **Index set.** An index set I is a set of ordered natural numbers where $|I| = t$. I is divided into l-subsets and t is a multiple of l: $I = \{I_1, I_2, \cdots, I_l\}$, where I_i denotes $i^{\text{-th}}$ element of I and consists of t/l indices i.e., $I_i = \{(t/l)(i-1)+1, (t/l)(i-1)+2, \cdots, (t/l)(i-1)+t/l\}$, for $1 \leq i \leq l$.

4.2. The Construction of Sec-DPoS. The Sec-DPoS scheme consists of four protocols. Firstly, we describe the key and index distribution protocol, where we assume that the client and management server communicate over a secure channel. The file upload process is divided into two protocols: the initial upload protocol and the subsequent upload protocol. Lastly, we describe the integrity auditing protocol.

4.2.1. Key and Index Distribution Protocol. An initial uploader generates an expected response set for a file to audit integrity using a message-derived key that is distributed from the management server. The expected response set is then uploaded with the file and every subsequent client who has the same file can use the expected response set generated by the first uploader. Note that every client who has the same file can get the same secret key via the management server. In this case, one important point is that used values in the expected response set should not be reused. However, for the case of a naïve solution, certain values can be reused as every client, including the first uploader, cannot know what values are used. In order to avoid the challenge index collision, we introduce a management server to preassign indices that are indicators of certain values in the expected response set. In addition, since clients encrypt files using the message-derived key that is distributed from the management server, outsourced data is resilient to dictionary attack in our model. The key and index distribution protocol is run as follows.

Step 1. Clt computes a hash value $h_F \longleftarrow H(F)$ and sends the key and index request to $Msrv$ with h_F.

Step 2. Upon receiving the key and index request, $Msrv$ invokes Algorithm 1 and sends outputs of Algorithm 1 to Clt.

Algorithm 1 generates a secret key, the highest challenge and an index subset. Upon receiving h_F, the management server checks whether h_F is already in data table T. If not, the management server computes $K_{h_F} \longleftarrow H(MK_{Msrv} \parallel h_F)$ using master key MK_{Msrv} and sets $\tau_{h_F} \longleftarrow 0$, $I^{h_F} \longleftarrow I$, where I is an initial ordered index set with $|I| = t$ (we assume that the size of the expected response set t is predetermined and publicly known) and τ_{h_F} (in our scheme, in order to audit integrity, the client precomputes the expected response set that contains a number of expected responses. That is,

Input : a ciphertext CT_F, a file tag value Tag_F, a secret key K_{h_F}, the highest challenge τ_{h_F}
Output : an expected response array $ResArr_{Clt}$
(1) for i in $[1, 3]$
(2) $K_{int,i} \longleftarrow KDF(K_{h_F}, i)$
(3) for j in $[1, t]$
(4) $ctr \longleftarrow \tau_{h_F} + j$
(5) $k \longleftarrow f_{K_{int,1}}(ctr)$
(6) $c \longleftarrow f_{K_{int,2}}(ctr||Tag_F)$
(7) $\sigma \longleftarrow CT_F[\pi_k(1)]||\cdots||CT_F[\pi_k(r)]$
(8) $v \longleftarrow H(c||\sigma)$
(9) $ResArr_{Clt}[ctr] \longleftarrow AE_{K_{int,3}}(v)$
(10) return $ResArr_{Clt}$

Algorithm 2: Precomputation process for integrity auditing.

Input : a ciphertext CT_F, a file tag value Tag_F, a master key MK_{Srv}, the highest challenge τ_{Tag_F}
Output : an expected response array $ResArr_{Srv}$
(1) for i in $[1, 2]$
(2) $K_{own,i} \longleftarrow KDF(MK_{Srv}, i||Tag_F)$
(3) for j in $[1, s]$
(4) $ctr \longleftarrow \tau_{Tag_F} + j$
(5) $k \longleftarrow f_{K_{own,1}}(ctr)$
(6) $c \longleftarrow f_{K_{own,2}}(ctr||Tag_F)$
(7) $\sigma \longleftarrow CT_F[\pi_k(1)]||\cdots||CT_F[\pi_k(d)]$
(8) $ResArr_{Srv}[ctr] \longleftarrow H(c||\sigma)$
(9) return $ResArr_{Srv}$

Algorithm 3: Precomputation process for ownership check.

the client can audit integrity as much as the number of precomputed responses. If all are used or assigned (i.e., $I^{h_F} = \emptyset$), a client has to generates a new expected response set. In this case, the cloud server can hold multiple expected response sets. The challenge index can then become confused. Thus, we use the highest challenge τ_{h_F} as a track of the challenge index in order to prevent confusion) is the highest challenge. The management server then records new data in T, where h_F is saved as the lookup key and K_{h_F}, τ_{h_F}, I^{h_F} are saved as values corresponding to h_F (line (1)-line (5)). If h_F is already in T, the management server loads values corresponding to h_F from T. After values are built or loaded (line (6)-line (10)), the management server randomly chooses an element I_α from I^{h_F} (if I^{h_F} is empty, then renewal I^{h_F} to I) and sends K_{h_F}, τ_{h_F}, I_α to client. If I^{h_F} was empty, the management server sends K_{h_F}, τ_{h_F}, I_α to the client with the expected response renewal request (line (11)-line (12)).

At the end of the key and index distribution protocol, the client receives K_{h_F},τ_{h_F}, I_α and the management server removes I_α in I^{h_F} (i.e., $I^{h_F} \longleftarrow I^{h_F} \setminus I_\alpha$).

4.2.2. Initial Upload Protocol. The initial upload process assumes that a file is uploaded as new data that is not previously been uploaded. Thus, the client generates the expected response set that is to be used in the integrity auditing and the cloud server generates another expected response set that is to be used in the ownership check. The initial upload protocol is run as follows.

Step 1. Clt generates an encryption key $K_F \longleftarrow KDF(K_{h_F}, 0)$ and computes $CT_F \longleftarrow Enc_{K_F}(F)$ and $Tag_F \longleftarrow H(CT_F)$. Clt then sends a file upload request to Srv with Tag_F (Tag_F is used as a file identifier).

Step 2. Upon receiving the upload request with Tag_F, Srv checks whether CT_F is in the storage using Tag_F. If not, Srv sends a data transmission request to Clt.

Step 3. Upon receiving the data transmission request, Clt generates an expected response set by invoking Algorithm 2 and sends encrypted data CT_F with expected response set $ResArr_{clt}$ to Srv.

Step 4. Upon receiving CT_F and $ResArr_{clt}$, Srv generates an expected response set by invoking Algorithm 3. Srv then stores CT_F at a secondary storage and keeps $ResArr_{Clt}$ and $ResArr_{Srv}$ in local storage.

Algorithm 2 generates an expected response set that contains t expected responses to be used in integrity auditing. Algorithm 2 takes encrypted data CT_F, a file tag value Tag_F, a secret key K_{h_F} and the highest challenge τ_{h_F} as an input and outputs an expected response set $ResArr_{clt}$.

Firstly, the client generates $K_{int,i} \longleftarrow KDF(K_{h_F}, i)$ for $1 \le i \le 3$ (line (1)-line (2)) and makes a counter ctr (line (4)). The client then generates a pseudorandom key k and nonce c corresponding to the counter ctr (line (5)-line (6)). Subsequently, the client extracts indices $\pi_k(1), \ldots, \pi_k(r)$ and generates a token σ by concatenating r bits within the encrypted data CT_F corresponding to the extracted indices, where $1 \le \pi_k(\cdot) \le |CT_F|$, $r \ll |CT_F|$ ($|CT_F|$ denotes the bit size of CT_F and $CT_F[l]$ denotes l-th bit of CT_F) (line (7)). The client then generates an expected response v by computing a hash value of σ with c (line (8)). Finally, the client encrypts v via the authenticated encryption scheme (line (9)). This procedure is repeated t times while increasing the counter ctr (line (3)-line (9)).

Algorithm 3 generates an expected response set that contains s expected responses to be used in the ownership check. Algorithm 3 takes a ciphertext CT_F, a file tag value Tag_F, a master key MK_{Srv} and the highest challenge τ_{Tag_F} as an input and outputs an expected response set $ResArr_{Srv}$. Note that the cloud server also uses the highest challenge as a track of the challenge index in the ownership check protocol. In the initial upload protocol, $\tau_{Tag_F} = 0$. The rest of the algorithm follows Algorithm 2. However, Algorithm 3 does not encrypt expected responses as $ResArr_{Srv}$ is retained and only used by the cloud server.

At the end of the initial upload protocol, the cloud server sets $Idx_{Srv} = 1$. Finally, the client holds $(K_{h_F}, Tag_F, \tau_{h_F}, h_F, I_\alpha)$ in local storage and the cloud server

Input : $Clt(CT_F)$, Srv (MK_{Srv}, Idx_{Srv}, Tag_F, $ResArr_{Srv}$)
Output : "*accept*" or "*reject*"

(1) ***Srv***: for i in [1, 2]
(2) $K_{own,i} \longleftarrow KDF(MK_{Srv}, i||Tag_F)$
(3) $\iota \longleftarrow \tau_{Tag_F} + Idx_{Srv}$
(4) $k \longleftarrow f_{K_{own,1}}(\iota)$
(5) $c \longleftarrow f_{K_{own,2}}(\iota||Tag_F)$
(6) send k, c to Clt as a challenge
(7) ***Clt***: compute $\sigma \longleftarrow CT_F[\pi_k(1)]||\cdots||CT_F[\pi_k(d)]$
(8) $P \longleftarrow H(c||\sigma)$
(9) send P to Srv as a response
(10) ***Srv***: if $P = ResArr_{Srv}[\iota]$
(11) return "*accept*"
(12) otherwise
(13) return "*reject*"

Algorithm 4: Ownership check protocol.

Input : $Clt(K_{h_F}, \tau_{h_F}, Idx_{Clt})$, $Srv(CT_F, ResArr_{Clt})$
Output : "*accept*" or "*reject*"

(1) ***Clt***: for i in [1, 3]
(2) $K_{int,i} \longleftarrow KDF(K_{h_F}, i)$
(3) $\iota \longleftarrow \tau_{h_F} + Idx_{Clt}$
(4) $k \longleftarrow f_{K_{int,1}}(\iota)$
(5) $c \longleftarrow f_{K_{int,2}}(\iota||Tag_F)$
(6) send k, c, ι to Srv as a challenge
(7) ***Srv***: compute $\sigma \longleftarrow CT_F[\pi_k(1)]||\cdots||CT_F[\pi_k(r)]$
(8) $P \longleftarrow H(c||\sigma)$
(9) send P, $ResArr_{Clt}[\iota]$ to Clt as a response
(10) ***Clt***: compute $v = AE^{-1}_{K_{int,3}}(ResArr_{Clt}[\iota])$
(11) if v is not valid
(12) return "*reject*"
(13) if $P = v$
(14) return "*accept*"
(15) otherwise
(16) return "*reject*"

Algorithm 5: Integrity auditing protocol.

holds $(MK_{Srv}, Tag_F, \tau_{Tag_F}, Idx_{Srv}, ResArr_{Clt}, ResArr_{Srv})$ in local storage and saves CT_F at the secondary storage. Note that all values that are stored in local storage have negligible size compared to the file.

4.2.3. Deduplication Protocol. The deduplication process assumes that a file is uploaded as duplicated data from a previous upload. Thus, the cloud server must verify that the client actually has the file. The deduplication protocol contains the ownership check protocol and is run as follows:

Step 1. Clt generates an encryption key $K_F \longleftarrow KDF(K_{h_F}, 0)$ and computes $CT_F \longleftarrow Enc_{K_F}(F)$ and $Tag_F \longleftarrow H(CT_F)$. Clt then sends a file upload request to Srv with Tag_F.

Step 2. Upon receiving the upload request, Srv checks whether CT_F is in the storage using Tag_F. If CT_F is in the storage, Srv runs Algorithm 4 with Clt.

Step 3. If Algorithm 4 returns "*accept*", Srv assigns a link of the file CT_F to the client. Otherwise, Srv returns "*reject*".

In Algorithm 4, the cloud server interacts with the client to verify that the client actually has the file. Firstly, the cloud server computes $K_{own,i} \longleftarrow KDF(MK_{Srv}, i \parallel Tag_F)$ for $1 \le i \le 2$ (line (1) ~ line (2)), where MK_{Srv} is a master secret key of the cloud server. The cloud server then generates the challenge $k = f_{K_{own,1}}(\iota)$ and $c = f_{K_{own,2}}(\iota||Tag_F)$, and sends them to the client, where $\iota = \tau_{Tag_F} + Idx_{Srv}$ (line (3)-line (6)). Upon receiving the challenge, the client extracts indices $\pi_k(1), \ldots, \pi_k(d)$ and generates a token σ by concatenating d bits within the ciphertext CT_F corresponding to the extracted indices (line (7)). The client then generates a proof P by hashing the token σ with nonce c (line (8)-line (9)). Finally, the client sends the proof to the cloud server as a response. Upon receiving the response, the cloud server verifies the response by comparing with the $\iota^{\text{-th}}$ value of the expected response set $ResArr_{Srv}$. If the proof P is equal to $RespArr_{Srv}[\iota]$, the cloud server accepts that the client actually has the file. Otherwise, the cloud server returns "reject" (line (10)-line (13)).

If ownership is accepted, the cloud server assigns a link of the file to the client. Thus, the client does not need to send the file. Note that, as every client who has the same file can get the same secret key, these clients can audit the file integrity without the need to upload any information.

At the end of the ownership check protocol, the cloud server computes $Idx_{Srv} = Idx_{Srv} + 1$ and if $Idx_{Srv} \equiv 0 \mod s$, and subsequently has to renew the expected response set $ResArr_{Srv}$. The renewal process is equal to Algorithm 3, except that the highest challenge $\tau_{Tag_F} = \tau_{Tag_F} + s$, where s is the size of expected response set.

4.2.4. Integrity Auditing Protocol. The client that has ownership of a file F can audit the integrity of the outsourced data F at any time. Before running the integrity auditing protocol, the client chooses one element from the assigned index set I_α, sets it to the challenge index Idx_{Clt} and removes the element from I_α.

Algorithm 5 presents the integrity auditing protocol. Firstly, the client generates $K_{int,i} \longleftarrow KDF(K_{h_F}, i)$ for $1 \le i \le 3$ (line (1)-line (2)). Then, the client computes $k = f_{K_{int,1}}(\iota)$ and $c = f_{K_{int,2}}(\iota \parallel Tag_F)$, where $\iota = \tau_{h_F} + Idx_{Clt}$ and sends k, c, ι to the cloud server as a challenge (line (3)-line (6)). Upon receiving the challenge, the cloud server extracts indices $\pi_k(1), \ldots, \pi_k(r)$ and generates a token σ by concatenating r bits within the ciphertext CT_F corresponding to the extracted indices (line (7)). The cloud server then generates a proof P by hashing the token σ with challenged nonce c (line (8)). Finally, the cloud server sends the proof P with the $\iota^{\text{-th}}$ value of the expected response set $ResArr_{Clt}$ to the client (line (9)). Upon receiving the proof with $ResArr_{Clt}[\iota]$, the client extracts an expected response value v by computing

$AE^{-1}_{K_{int,3}}(ResArr_{Clt}[\iota])$ (line (10)). If v is not valid, the client returns "reject" (line (11)-line (12)). Otherwise the client compares the proof P with v. If P equals v, the client accepts that the outsourced data is intactly stored. Otherwise, the client returns "reject" (line (13)-line (16)).

At the end of the integrity auditing protocol, if all the preassigned challenge index are used (i.e., $I_\alpha = \emptyset$), the client has to obtain a new challenge index set from the management server. The challenge index reissuing process is similar to Algorithm 1, however the management server does not need to load or send the secret key. Note that if the client has to renew the expected response set, the client runs Algorithm 2 and sends a new expected response set to the cloud server.

5. Security Analysis

In this section, we analyze the security of Sec-DPoS. Firstly, we formalize the security definitions that consist of two parts: *client uncheatability* in proof of ownership and *server unforgeability* in integrity auditing.

5.1. Security Definitions. In the cross-user client-side deduplication system, a malicious client that has only partial information of a file F can attempt to convince the cloud server in the ownership check protocol. Thus, it is necessary that the malicious clients cannot cheat the cloud server for ownership of the entire file. We first summarize the overall process of the ownership check protocol in our Sec-DPoS scheme.

When a client attempts to upload the duplicated data, the cloud server generates a random seed key k with nonce c and sends these to the client as a challenge. Upon receiving the challenge, the client extracts indices $\pi_k(1), \ldots, \pi_k(d)$ and generates a token σ by concatenating d bits within the ciphertext CT_F corresponding to the extracted indices. Subsequently, the client generates a proof by hashing the token σ with nonce c and sends the proof to the cloud server as a response. Finally, the protocol outputs "accept" if the proof is valid. Otherwise, the protocol returns "reject".

Now, we present the definition of client uncheatability over Sec-DPoS, based on game scenario between a challenger C (the role of the cloud server) and an adversary A (the role of the client). We prove the security of Sec-DPoS over a weak assumption, that the adversary can build an expected token σ' before the challenger makes a challenge. This means that the adversary can get d bits of ciphertext via an oracle, even if the adversary does not have the whole file.

In the Sec-DPoS scheme, the experiment $Exp^{uncheat}_{A,Sec-DPoS}$ for uncheatability can be described as follows:

(i) **Setup phase**. The challenger C randomly chooses a data D, a master key $MK \longleftarrow \{0,1\}^\lambda$ and sends D to an oracle. C then sets up the ownership check system of Sec-DPoS over D.

(ii) **Learning phase**. The adversary A can query the oracle at any point in time with d indices. If queried, the oracle replies all bits corresponding to queried indices from D.

(iii) **Challenge phase**. C run the ownership check protocol with A. If the ownership check protocol returns "accept", $Exp^{uncheat}_{A,Sec-DPoS}$ outputs 1. Otherwise, $Exp^{uncheat}_{A,Sec-DPoS}$ outputs 0.

Definition 1 (client uncheatability). The Sec-DPoS scheme is uncheatable if for any probabilistic polynomial time (PPT) adversary A and for security parameter λ,

$$\Pr\left[Exp^{uncheat}_{A,Sec-DPoS}\left(1^\lambda\right) = 1\right] \leq \epsilon(\lambda), \tag{1}$$

where $\epsilon(\cdot)$ is a negligible function.

Next, we have to consider server unforgeability. The client that has the ownership of a file F can audit integrity of the outsourced data at any time. In our Sec-DPoS scheme, the integrity auditing protocol is similar to the ownership check protocol, however the role of prover and verifier is changed and the cloud server take t expected responses (in encrypted form) provided by the client. According to this situation, we can build the security game.

The definition of server unforgeability is defined based on the game scenario between a challenger C (the role of the client) and an adversary A (the role of the cloud server). Then, the experiment $Exp^{unforge}_{A,Sec-DPoS}$ for unforgeability is described as follows:

(i) **Setup phase**. The challenger C randomly chooses a data D, a secret key $K \longleftarrow \{0,1\}^\lambda$ and sends D to an oracle. C then sets up the integrity auditing system of Sec-DPoS over D.

(ii) **Query phase**. The adversary A can query the oracle at any point in time with r indices. If queried, the oracle replies all bits corresponding to the queried indices from D. If the adversary A queries the challenger C, C returns an expected response in encrypted form.

(iii) **Challenge phase**. C run the integrity auditing protocol with A. If the integrity auditing protocol returns "accept", $Exp^{unforge}_{A,Sec-DPoS}$ outputs 1. Otherwise, $Exp^{unforge}_{A,Sec-DPoS}$ outputs 0.

Definition 2 (server unforgeability). The Sec-DPoS scheme is unforgeable if for any PPT adversary A and for security parameter λ,

$$\Pr\left[Exp^{unforge}_{A,Sec-DPoS}\left(1^\lambda\right) = 1\right] \leq \theta(\lambda), \tag{2}$$

where $\theta(\cdot)$ is a negligible function.

5.2. Security Proof. In this subsection, we prove the security of Sec-DPoS corresponds to the security definition.

Before we prove the security in terms of the security definition, we first have to go through the data confidentiality of Sec-DPoS. Regarding the data confidentiality in our model, we argue the following theorem.

Theorem 3. *Sec-DPoS ensures confidentiality with a brute-force attack resilience if any PPT adversary is not allowed to compromise with the management server.*

Proof. In our system, a management server generates the convergent key associated with a private key of the management server and we assume that the key distribution process is run over a secure channel. Thus, no adversary who attempts to launch a brute force attack can generate the valid encryption key without the private key of the management server. Therefore, even if the file is predictable, the adversary cannot guess the plaintext by launching a brute force attack.

However, the adversary can attempt a brute force attack via the management server as the management server cannot make a distinction between an honest and malicious client. Hence, by applying a per-client or per-file limitation strategy, as in [13] or [15], we can achieve confidentiality with a brute force attack resilience.

Now, we prove the security for uncheatability and unforgeability under the assumption that Theorem 3 holds. □

Theorem 4. *Let λ be a security parameter. Let H be a random oracle with output length $\delta(\lambda)$. Assume that the pseudorandom function f and the pseudorandom permutation π are secure with key length $\gamma(\lambda)$. Sec-DPoS holds client uncheatability for any PPT adversary who can make $q_H(\lambda)$ queries to H and $q_O(\lambda)$ queries to an oracle that returns a corresponding bit from ciphertext to queried index.*

Proof. To show that Theorem 4 holds, we consider the experiment $Exp^{uncheat}_{A,Sec-DPoS}$ in Definition 1. In our experiment $Exp^{uncheat}_{A,Sec-DPoS}$, the adversary can get $d \cdot q_O$ bits of the data D via the oracle, where $q_O = q_O(\lambda)$, and the challenger runs the ownership check protocol with the adversary. The advantage that the adversary wins the uncheatability game is:

$$\Pr\left[Exp^{uncheat}_{A,Sec-DPoS}\left(1^{\lambda}\right) = 1\right]. \tag{3}$$

□

The second experiment $Exp^{uncheat,2}_{A,Sec-DPoS}$ is identical to $Exp^{uncheat}_{A,Sec-DPoS}$ except that the adversary encounters a hash collision. Since H is a random oracle, we have

$$\begin{aligned}&\Pr\left[\text{the adversary encounters the hash collision}\right]\\ &\quad\le \frac{q_H}{2^{\delta}},\end{aligned} \tag{4}$$

where $q_H = q_H(\lambda)$ and $\delta = \delta(\lambda)$. Then, we can have

$$\begin{aligned}&\Pr\left[Exp^{uncheat}_{A,Sec-DPoS}\left(1^{\lambda}\right) = 1\right]\\ &\quad\le \Pr\left[Exp^{uncheat,2}_{A,Sec-DPoS}\left(1^{\lambda}\right) = 1\right] + \frac{q_H}{2^{\delta}}.\end{aligned} \tag{5}$$

The third experiment $Exp^{uncheat,3}_{A,Sec-DPoS}$ is identical to $Exp^{uncheat,2}_{A,Sec-DPoS}$ except that the adversary predicts the random seed key k and nonce c that are to be challenged. Since we assume that the pseudorandom function f is secure, we have

$$\Pr\left[\text{the adversary predict } k \text{ and } c\right] \le \frac{1}{2^{\gamma}}, \tag{6}$$

where $\gamma = \gamma(\lambda)$. Then, we can have

$$\begin{aligned}&\Pr\left[Exp^{uncheat}_{A,Sec-DPoS}\left(1^{\lambda}\right) = 1\right]\\ &\quad\le \Pr\left[Exp^{uncheat,3}_{A,Sec-DPoS}\left(1^{\lambda}\right) = 1\right] + \frac{q_H}{2^{\delta}} + \frac{1}{2^{\gamma}}.\end{aligned} \tag{7}$$

The advantage of the third experiment can be determined as follows.

Suppose that the adversary can obtain at most a l fraction of D(i.e., $l = (d \cdot q_O)/|D|$). Now, we compute the probability that an adversary who can get a l fraction of D will pass the ownership checking protocol.

In the ownership checking protocol of our proposed scheme, the client has to extract d bits from the target data D, corresponding to d random indices. Let ω be an event that the adversary owns l fraction of D, and $pass_1$ be an event that the adversary successfully passes a single-bit challenge in $Exp^{uncheat,3}_{A,Sec-DPoS}$ (i.e., $d = 1$). The probability $\Pr(pass_1)$ can then be computed as follows:

$$\begin{aligned}\Pr\left(pass_1\right) &= \Pr\left(pass_1 \wedge (\omega \vee \overline{\omega})\right)\\ &= \Pr\left(pass_1 \wedge \omega\right) \vee \Pr\left(pass_1 \wedge \overline{\omega}\right)\\ &= \Pr\left(pass_1 \mid \omega\right) P\left(\omega\right)\\ &\quad + \Pr\left(pass_1 \mid \overline{\omega}\right) \Pr\left(\overline{\omega}\right).\end{aligned} \tag{8}$$

If the single-bit challenge is in the known fraction then the adversary can always pass (i.e., $\Pr(pass_1 \mid \omega) = 1$). However, the adversary cannot response correctly if the single-bit challenge is in the unknown fraction. In this case, the best way of successful pass is to guess the response, i.e., $\Pr(pass_1 \mid \overline{\omega}) = 0.5 + negligible$. Let $\Pr(pass_1 \mid \overline{\omega}) \le g$. Then, we have

$$\begin{aligned}&\Pr\left(pass_1 \mid \overline{\omega}\right) P\left(\omega\right) + \Pr\left(pass_1 \mid \overline{\omega}\right) P\left(\overline{\omega}\right)\\ &\quad\le 1 \cdot l + g \cdot (1 - l).\end{aligned} \tag{9}$$

Finally, we can compute the probability $\Pr(pass)$ that the adversary successfully passes a d bits challenge by

$$\Pr\left(pass\right) = \left\{\Pr\left(pass_1\right)\right\}^d \le \left\{l + g \cdot (1 - l)\right\}^d. \tag{10}$$

We desire a probability $\Pr(pass) \le 2^{-\mu(\lambda)}$. To achieve client uncheatability, the size of the challenge d can be derived as follows:

$$\begin{aligned}&\left\{l + g \cdot (1 - l)\right\}^d \le 2^{-\mu}.\\ &\qquad\therefore d \ge \frac{\mu \cdot \ln 2}{1 - \left(l + g\left(1 - l\right)\right)},\end{aligned} \tag{11}$$

where $\mu = \mu(\lambda)$. Thus, if the size of challenge $d = \lceil(\mu \cdot \ln 2)/(1 - (l + g(1 - l)))\rceil$, we can have

$$\Pr\left[Exp^{uncheat,3}_{A,Sec-DPoS}\left(1^{\lambda}\right) = 1\right] \le \frac{1}{2^{\mu}}. \tag{12}$$

Then, we can show that Sec-DPoS holds client uncheatability as follows:

$$\Pr\left[Exp^{uncheat}_{A,Sec-DPoS} = 1\right] \le \frac{1}{2^{\mu}} + \frac{q_H}{2^{\delta}} + \frac{1}{2^{\gamma}}. \tag{13}$$

Additionally, as a realistic scenario, the adversary may know part of the plaintext. Thus, we also have to consider the scenario whereby the adversary who has a fraction of plaintext attempts to convince the challenger. Let ω' be an event that the adversary owns β fraction of plaintext and $pass_1'$ be an event that the adversary successfully passes a single-bit challenge in the ownership check protocol (i.e., $d = 1$). The probability $\Pr(pass_1')$ can then be computed as follows:

$$\begin{aligned}\Pr\left(pass_1'\right) &= \Pr\left(pass_1' \wedge \left(\omega' \vee \overline{\omega'}\right)\right)\\ &= \Pr\left(pass_1' \wedge \omega'\right) \vee \Pr\left(pass_1' \wedge \overline{\omega'}\right)\\ &= \Pr\left(pass_1' \mid \omega'\right) P\left(\omega'\right)\\ &\quad + \Pr\left(pass_1' \mid \overline{\omega'}\right) \Pr\left(\overline{\omega'}\right).\end{aligned} \tag{14}$$

Under the assumption that Theorem 3 holds, unlike $Exp^{uncheat,3}_{A,Sec-DPoS}$, the adversary has to guess the response regardless of the challenged position (i.e., $\Pr(pass_1' \mid \omega') = \Pr(pass_1' \mid \overline{\omega'}) = 0.5 + negligible$). Thus, Let $\Pr(pass_1' \mid \omega') = \Pr(pass_1' \mid \overline{\omega'}) \le g$, then

$$\begin{aligned}&\Pr\left(pass_1' \mid \omega'\right) P\left(\omega'\right) + \Pr\left(pass_1' \mid \overline{\omega'}\right) \Pr\left(\overline{\omega'}\right)\\ &\quad \le g \cdot \beta + g \cdot (1 - \beta) = g.\end{aligned} \tag{15}$$

Finally, we can compute probability $\Pr(pass')$ that the adversary successfully passes a d bit challenge by

$$\Pr\left(pass'\right) = \left\{\Pr\left(pass_1'\right)\right\}^d \le g^d. \tag{16}$$

We desire a probability $\Pr(pass') \le 2^{-\mu(\lambda)}$. To achieve client uncheatability in this scenario, the size of the challenge d can be derived as follows:

$$\begin{aligned}&g^d \le 2^{-\mu}.\\ &\therefore d = \left\lceil \frac{\mu \cdot \ln 2}{1 - g} \right\rceil,\end{aligned} \tag{17}$$

where $\mu = \mu(\lambda)$. Thus, we can take the size of the challenge as $d = \max(\lceil(\mu \cdot \ln 2)/(1 - (l + g(1 - l)))\rceil, \lceil(\mu \cdot \ln 2)/(1 - g)\rceil)$. However, since $\lceil(\mu \cdot \ln 2)/(1-g)\rceil < \lceil(\mu \cdot \ln 2)/(1-(l+g(1-l)))\rceil$, we have $d = \lceil(\mu \cdot \ln 2)/(1 - (l + g(1 - l)))\rceil$.

Next, we can directly have the following theorem for server unforgeability.

Theorem 5. *Let λ be a security parameter. Let H be a random oracle with output length $\delta(\lambda)$. Assume that the pseudorandom function f and the pseudorandom permutation π are secure with key length $\gamma(\lambda)$. If AE is an ideal authenticated encryption function with key length $\rho(\lambda)$ and authentication code length $\varphi(\lambda)$, Sec-DPoS holds server unforgeability for any PPT adversary who can make $q_H(\lambda)$ queries to H and $q_O(\lambda)$ queries to an oracle that returns a corresponding bit from ciphertext to queried index.*

Proof. To show that Theorem 5 holds, we consider the experiment $Exp^{unforge}_{A,Sec-DPoS}$ in Definition 2. In our experiment $Exp^{uncheat}_{A,Sec-DPoS}$, the adversary can get $r \cdot q_O$ bits of the data D via the oracle and can get t expected responses by the authenticated encryption form, where $q_O = q_O(\lambda)$. The challenger then runs the integrity auditing protocol with the adversary. The advantage that the adversary wins the unforgeability game is

$$\Pr\left[Exp^{unforge}_{A,Sec-DPoS}\left(1^\lambda\right) = 1\right]. \tag{18}$$

□

The second experiment $Exp^{unforge,2}_{A,Sec-DPoS}$ is identical to $Exp^{unforge}_{A,Sec-DPoS}$ except that the adversary attempts to cheat the authenticated encryption function. In this case, the adversary attempts to break encryption or forge the authentication code. In the Sec-DPoS, since we assume that AE is an ideal authenticated encryption function, we have

$$\begin{aligned}&\Pr\left[\text{the adversary cheats authenticated encryption}\right]\\ &\quad \le \frac{1}{2^\rho} + \frac{t}{2^\varphi},\end{aligned} \tag{19}$$

where $\rho = \rho(\lambda)$ and $\varphi = \varphi(\lambda)$. Then, we have

$$\begin{aligned}&\Pr\left[Exp^{unforge}_{A,Sec-DPoS}\left(1^\lambda\right) = 1\right]\\ &\quad \le \Pr\left[Exp^{unforge,2}_{A,Sec-DPoS}\left(1^\lambda\right) = 1\right] + \frac{1}{2^\rho} + \frac{t}{2^\varphi}.\end{aligned} \tag{20}$$

The third experiment $Exp^{unforge,3}_{A,Sec-DPoS}$ is identical to $Exp^{unforge,2}_{A,Sec-DPoS}$ except that the adversary encounters a hash collision. Since H is a random oracle, we have

$$\begin{aligned}&\Pr\left[\text{the adversary encounters the hash collision}\right]\\ &\quad \le \frac{q_H}{2^\delta},\end{aligned} \tag{21}$$

where $q_H = q_H(\lambda)$ and $\delta = \delta(\lambda)$. Then, we can have

$$\begin{aligned}&\Pr\left[Exp^{unforge}_{A,Sec-DPoS}\left(1^\lambda\right) = 1\right]\\ &\quad \le \Pr\left[Exp^{unforge,3}_{A,Sec-DPoS}\left(1^\lambda\right) = 1\right] + \frac{q_H}{2^\delta} + \frac{1}{2^\rho} + \frac{t}{2^\varphi}.\end{aligned} \tag{22}$$

The forth experiment $Exp^{unforge,4}_{A,Sec-DPoS}$ is identical to $Exp^{unforge,3}_{A,Sec-DPoS}$ except that the adversary predicts the random seed key k and nonce c that are to be challenged. Since we assume that pseudorandom function f is secure, we have

$$\Pr\left[\text{the adversary predict } k \text{ and } c\right] \le \frac{1}{2^\gamma}, \tag{23}$$

where $\gamma = \gamma(\lambda)$. Then, we can have

$$\begin{aligned}&\Pr\left[Exp^{unforge}_{A,Sec-DPoS}\left(1^\lambda\right) = 1\right]\\ &\quad \le \Pr\left[Exp^{unforge,4}_{A,Sec-DPoS}\left(1^\lambda\right) = 1\right] + \frac{q_H}{2^\delta} + \frac{1}{2^\gamma} + \frac{1}{2^\rho}\\ &\qquad + \frac{t}{2^\varphi}.\end{aligned} \tag{24}$$

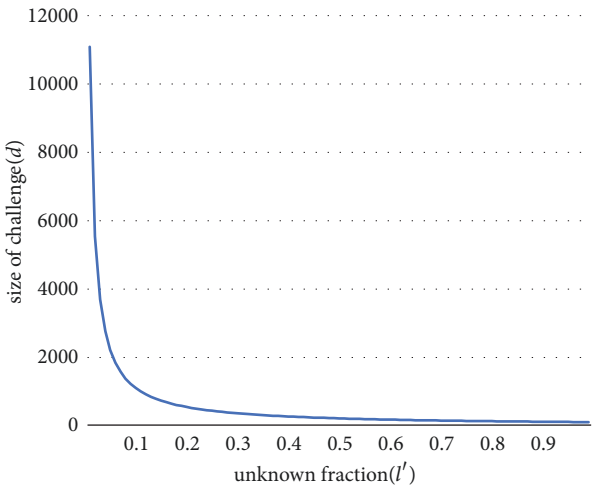

Figure 2: Number of challenging bits for 80-bit security in the ownership check protocol.

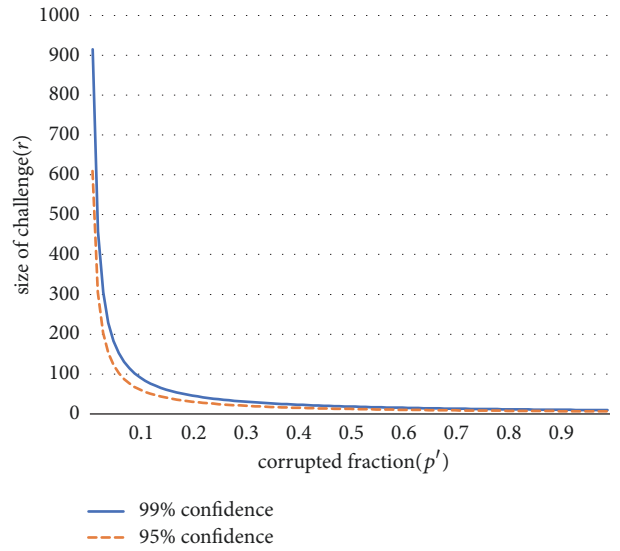

Figure 3: Number of challenging bits for fixed confidence in the integrity auditing protocol.

The advantage of the forth experiment can be determined as follows.

Suppose that the adversary can obtain at most a p fraction of D (i.e., $p = (r \cdot q_O)/|D|$). Now, we compute the probability that an adversary who can get a p fraction of D can pass the integrity auditing protocol.

In the integrity auditing protocol of our proposed scheme, the cloud server has to extract r bits from the target data D corresponding to r random indices. The rest is similar to the proof in $Exp^{uncheat,3}_{A,Sec-DPoS}$. Thus, if the size of challenge $r = \lceil(\mu \cdot \ln 2)/(1 - (p + g(1 - p)))\rceil$, we can have

$$\Pr\left[Exp^{unforge,4}_{A,Sec-DPoS}\left(1^{\lambda}\right) = 1\right] \leq \frac{1}{2^{\mu}}, \tag{25}$$

where $\mu = \mu(\lambda)$. We can then show that Sec-DPoS holds client unforgeability as follows:

$$\Pr\left[Exp^{uncheat}_{A,Sec-DPoS} = 1\right] \leq \frac{1}{2^{\mu}} + \frac{q_H}{2^{\delta}} + \frac{1}{2^{\gamma}} + \frac{1}{2^{\rho}} + \frac{t}{2^{\varphi}}. \tag{26}$$

5.3. Analysis of Detection Probability. Unlike other schemes in [1, 2, 9–12], we apply a bit-level challenge to Sec-DPoS. Thus, we should analyze the detection probability when an attacker who has only a fraction of data attempts to convince that it owns the whole data or the data is stored intactly in the ownership check protocol or integrity auditing protocol. In the integrity auditing and ownership check protocol, the size of the challenge is d and r, respectively, and we can choose these by setting the desired security parameter k. As proved above, $d = \lceil(\mu \cdot \ln 2)/|(1 - (l + g(1 - l)))|\rceil$, $r = \lceil(\mu \cdot \ln 2)/|(1 - (p + g(1 - p)))|\rceil$, where g denotes the guessing probability of single-bit challenge for an unknown fraction and t and p denote that adversary knows t and p fractions of the target data (we assume that adversary can get a portion of encrypted data by querying to an oracle) in the ownership check and integrity auditing protocol, respectively.

Firstly, we analyze the detection probability in the ownership check protocol. Let $l = (1 - l')$. Then we can have

$$d = \left\lceil \frac{\mu \cdot \ln 2}{|l'(1 - g)|} \right\rceil, \tag{27}$$

where l' denotes the unknown fraction of the target data that the client does not know. As shown in Figure 2, we present the size of the challenge d by setting l' to different values for various security parameters.

Next, we analyze the detection probability in the integrity auditing protocol. Let $p = (1 - p')$. Then we can have

$$r = \left\lceil \frac{\mu \cdot \ln 2}{|p'(1 - g)|} \right\rceil, \tag{28}$$

where p' denotes the unknown fraction of the target data that the cloud server does not know. Note that the unknown fraction in this case is the same as the corrupted fraction. This means that the cloud server cannot determine the damaged fraction when the data is corrupt from unintentional errors. As shown in Figure 3, we present the size of challenge r by setting p' differently for 99% and 95% confidence.

6. Implementation

In this section, we present the implementation results, evaluate our Sec-DPoS scheme, and compare with other schemes. In order to evaluate the efficiency of Sec-DPoS, we compared with other schemes, namely, SecCloud [1], Message-locked PoOR [11], and DeyPoS [12], and all schemes are implemented and evaluated over Intel Core i7-4790 CPU @ 3.60 GHz. All implementation results represent the median value of 100 trials.

6.1. Implementation of Ownership Check Protocol. The ownership check process in Sec-DPoS simply challenges d random

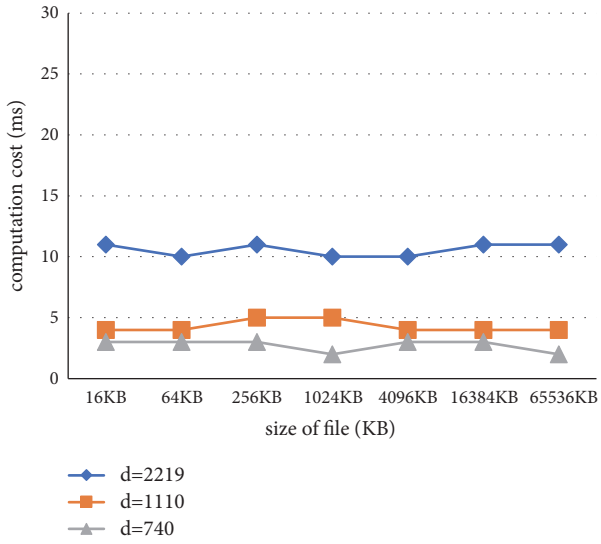

Figure 4: Computation costs for various file sizes for the ownership check protocol, when the number of challenged bits is 2219, 1110, and 740, respectively.

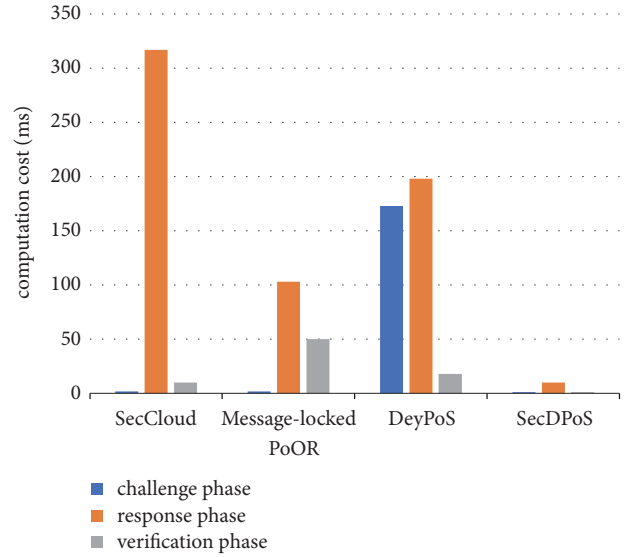

Figure 5: Comparison of the computation cost with other schemes for the ownership check protocol, when the file size is 64 MB.

indices for an encrypted file. As analyzed in Section 5, the size of the challenge can be set by setting l', which denotes the unknown fraction of the target data. Note that the ownership check protocol in Sec-DPoS ensures $(1 - 2^{-\mu})$ confidence. In our implementation, we take $\mu = 80$ and $l' \in \{0.05, 0.10, 0.15\}$. We assume that the selected l' is sufficiently reasonable as the adversary has to know 95%, 90% and 85% for encrypted data, not plaintext. As shown in Figure 2, we set the size of the challenge as $d \in \{2219, 1110, 740\}$ for $\mu = 80$ and $l' \in \{0.05, 0.10, 0.15\}$. We present the implementation results for various d and various sizes of data (see Figure 4). As shown in Figure 4, the time cost of the ownership check protocol in Sec-DPoS does not depend on the size of the data.

In order to evaluate the efficiency of Sec-DPoS for the ownership checking protocol, we also measured the challenge phase, the response phase, and the verification phase, respectively. As shown in Figure 5, we present our implementation results compared to other schemes for a 64 MB file. In particular, Sec-DPoS was measured when the size of the challenge is 2219 for 80-bit security. For the case of SecCloud, the same scheme was used as with PoWs [4]. For Message-locked PoOR, since the scheme is based on HMAC, the target file needs to be accessed in the ownership check process (other schemes do not access the target file). For the implementation of Message-locked PoOR, we set the size of the block as 4 KB, the same setting used in [11]. For DeyPoS, since the scheme is based on a homomorphic authenticated tree, there is a large variation in efficiency, depending on block size (if the size of the block is small, the height of the tree increases and vice versa). For the implementation of DeyPoS, we set the size of the block as 64 KB and the computation cost was measured more efficiently compared with SecCloud and Message-locked PoOR ([12] set the size of block as 4 KB, 16 KB, 64 KB, with the latter having the highest efficiency in our experiment). For the case of Sec-DPoS, the computation cost was measured as 0.1 ms, 10 ms and 0.001 ms in the challenge, response, and verification phase, respectively. Hence, we can evaluate that our Sec-DPoS scheme has the highest efficiency for the ownership check protocol. Note that since SecCloud and DeyPoS are based on the tree structure in the ownership check, if the size of the file increases, time cost increases while the time costs of Message-locked PoOR and Sec-DPoS are constant. In terms of the network latency, Sec-DPoS and Message-locked PoOR have $O(1)$ communication cost and the others need $O(b \log n)$ communication cost.

6.2. Implementation of Integrity Auditing Protocol. The integrity auditing process in Sec-DPoS simply challenges r random indices for an encrypted file. As analyzed in Section 5, the size of the challenge can be determined by setting p' to denote the corrupt fraction of the target data. In our implementation, we implement the integrity auditing protocol of Sec-DPoS for 99% and 95% confidence. We stress that 99% and 95% confidence are the same conditions as those of other schemes. As shown in Figure 3, we set the size of the challenge as $r = 930$ for 99% confidence and $r = 610$ for 95% confidence when the corrupted fraction of the target data is 0.01 (i.e., $p' = 1\%$). We present implementation results for $r = 930, 610$ and various sizes of data (see Figure 6). As shown in Figure 6, the time cost of the integrity auditing protocol in Sec-DPoS does not depend on the size of the file.

In order to evaluate the efficiency of Sec-DPoS for the integrity auditing protocol, we also measured the challenge phase, response phase, and verification phase, respectively. As shown in Figure 7, we present our implementation results compared with other schemes for a 64 MB file. All schemes were measured for 99% confidence. In the case of SecCloud, since the scheme is based on public key cryptography, the computation cost was measured to be greater than 300 ms. For Message-locked PoOR, we set the size of the block as 4 KB

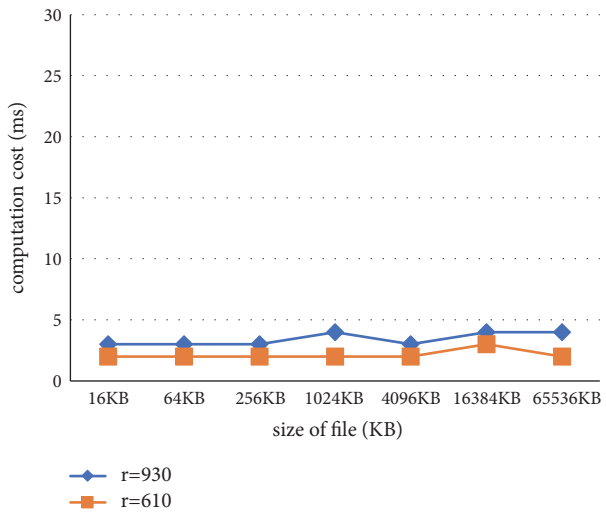

Figure 6: Computation costs for various file sizes for the integrity auditing protocol, when the number of challenged bits is 930 and 610, respectively.

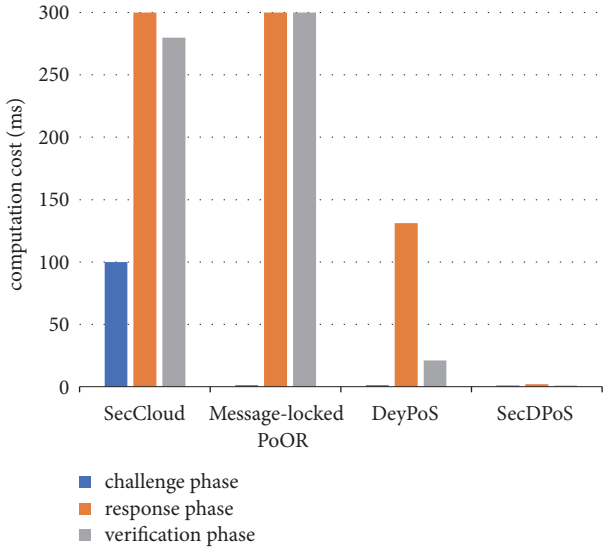

Figure 7: Comparison of computation costs with other schemes for the integrity auditing protocol, when the file size is 64 MB.

(the same setting as with [11]) and the computation cost was also measured to be greater than 300 ms. For case of DeyPoS, we set the size of the block as 64 KB and the computation cost was measured more efficiently ([12] set the size of the block as 4 KB, 16 KB, 64 KB, with the latter having the greatest efficiency in our experiment). However, since DeyPoS is based on a tree structure for the integrity auditing, if the size of the file increases, time cost also increases. Moreover, since the size of the block is set as 64 KB and the number of challenge is 480, the entire file has to be accessed with less than 30720 KB, which is impractical. Similarly, Message-locked PoOR also needs access to the entire file for less than 1920 KB. For the case of Sec-DPoS, the computation cost was measured 0.1 ms, 2 ms and 0.03 ms in the challenge,

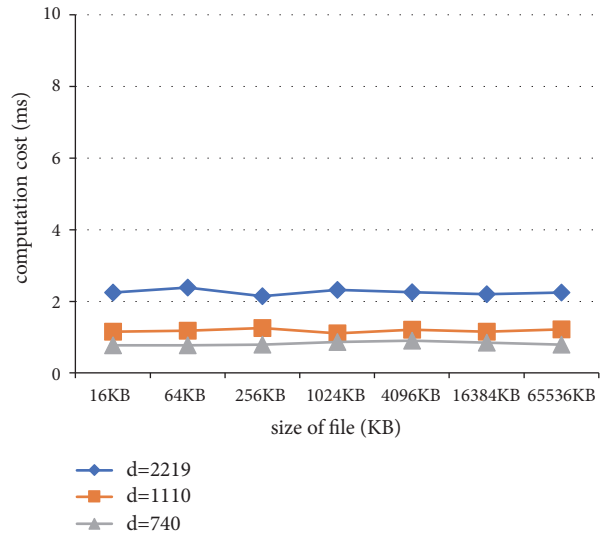

Figure 8: Computation costs for various file sizes in the initialization process for the ownership check, when the number of challenged bits is 2219, 1110, and 740, respectively.

response, and verification phase, respectively. Hence, we can evaluate that Sec-DPoS has the greatest efficiency for the integrity auditing protocol. In terms of the network latency, Sec-DPoS and Message-locked PoOR have $O(1)$ communication cost and the others need $O(b \log n)$ communication cost.

6.3. Implementation of the Initialization Phase. When the client uploads fresh data, the client and cloud server have to invoke a precomputation process. In the precomputation process, the client generates t expected responses for integrity auditing and the cloud server generates s expected responses for the ownership check. In addition, the size of challenge is r in the integrity auditing and d in the ownership check. Therefore, we measured the time cost of the initialization process by varying each variable for the integrity auditing and ownership check, respectively. In this experiment, we implemented over Intel Core i7-4790 CPU @ 3.60 GHz.

For the ownership check protocol, we measured time cost for various file sizes for $d \in \{2219, 1110, 740\}$ and $s = 1000$. As shown in Figure 8, the precomputation process for the ownership check does not depend on the file size and time cost was measured as constant. When $d = 2219, 1110$ and 740, the time cost was measured to be approximately 2.2 ms, 1.1 ms and 0.7ms, respectively. For the integrity auditing protocol, we also measured the time cost for various file size when we take $r \in \{930, 610\}$ and $t = 5000$. We assume that the integrity auditing protocol is executed more frequently than the ownership check protocol. Hence, we set t to be larger than s. As shown in Figure 9, the precomputation process for integrity auditing also does not depend on the file size and the time cost was measured as constant. When $r = 930$ and 610, the time cost was measured to be approximately 4.7 ms and 3.1 ms, respectively. Note that if the number of precomputed responses (i.e., s or t) increases, the time cost also increases linearly.

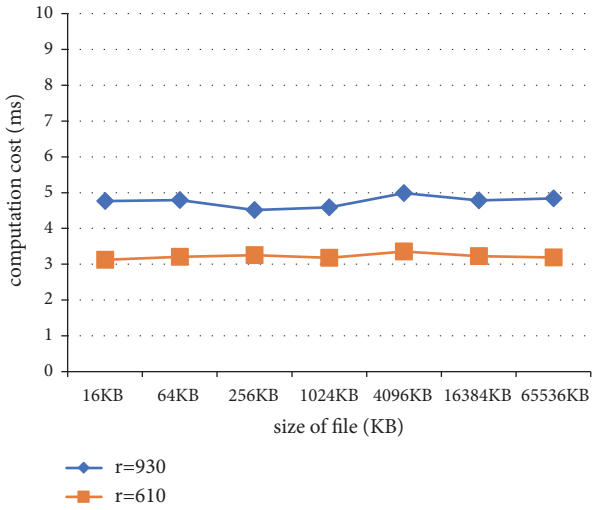

Figure 9: Computation costs for various file sizes in the initialization process for the integrity auditing, when the number of challenged bits is 930 and 610, respectively.

7. Conclusion

In order to comply with the three important requirements of cloud storage environments: data confidentiality, integrity, and storage efficiency, we proposed a secure and highly efficient Sec-DPoS scheme based on symmetric key cryptography. In our proposal, the scheme ensures data confidentiality with dictionary attack resilience and can efficiently audit integrity of outsourced data while the cloud server saves resources. By applying a bit-level challenge, we designed Sec-DPoS to perform efficiently, even for small data types. Moreover, we proved the security of Sec-DPoS in the random oracle model with the information theory, and experimental results show that Sec-DPoS has the highest efficiency compared with other schemes.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2016R1D1A1B03931071), Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government (MSIT) (No. B0717-16-0097, Development of V2X Service Integrated Security Technology for Autonomous Driving Vehicle), and Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean Government [18ZH1200, Core Technology Research on Trust Data Connectome].

References

[1] J. Li et al., "Secure auditing and deduplication data in cloud," *IEEE Transactions on Computers and Cloud Computing*, pp. 1–11, 2015.

[2] J. Yuan and S. Yu, "Secure and constant cost public cloud storage auditing with deduplication," in *Proceedings of the 1st IEEE International Conference on Communications and Network Security (CNS '13)*, October 2013.

[3] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Security & Privacy*, vol. 8, no. 6, pp. 40–47, 2010.

[4] S. Halevi et al., "Proofs of ownership in remote storage systems," in *Proceedings of the 18th ACM conference on Computer and communications security*, ACM, 2011.

[5] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *Proceedings of the 22nd International Conference on Distributed Systems*, IEEE, July 2002.

[6] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, Berlin, Germany, 2013.

[7] G. Ateniese, R. Burns, R. Curtmola et al., "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, ACM, November 2007.

[8] A. Juels and B. S. Kaliski Jr., "Pors: proofs of retrievability for large files," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, ACM, 2007.

[9] Q. Zheng and S. Xu, "Secure and efficient proof of storage with deduplication," in *Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy, CODASPY '12*, ACM, 2012.

[10] R. Du et al., "Proofs of ownership and retrievability in cloud storage," in *Proceedings of the 13th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom '14*, IEEE, September 2014.

[11] J. Chen, L. Zhang, K. He, M. Chen, R. Du, and L. Wang, "Message-locked proof of ownership and retrievability with remote repairing in cloud," *Security and Communication Networks*, vol. 9, no. 16, pp. 3452–3466, 2016.

[12] K. He, J. Chen, R. Du, Q. Wu, G. Xue, and X. Zhang, "DeyPoS: deduplicatable dynamic proof of storage for multi-user environments," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3631–3645, 2016.

[13] S. Keelveedhi, M. Bellare, and T. Ristenpart, "DupLESS: server-aided encryption for deduplicated storage," in *Proceedings of the 22nd USENIX Security Symposium*, 2013.

[14] R. Di Pietro and A. Sorniotti, "Boosting efficiency and security in proof of ownership for deduplication," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2012*, ACM, May 2012.

[15] J. Liu, N. Asokan, and B. Pinkas, "Secure deduplication of encrypted data without additional independent servers," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS 2015*, ACM, October 2015.

[16] H. Qi et al., "Secure data deduplication scheme based on distributed random key in integrated networks," in *Proceedings of the 2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, December 2017.

[17] M. Liu, C. Yang, Q. Jiang, X. Chen, J. Ma, and J. Ren, "Updatable block-level deduplication with dynamic ownership management on encrypted data," in *Proceedings of the 2018 IEEE International Conference on Communications (ICC '18)*, IEEE, May 2018.

[18] N. Kaaniche and M. Laurent, "A secure client side deduplication scheme in cloud storage environments," in *Proceedings of the 2014 6th International Conference on New Technologies, Mobility and Security, NTMS '14*, April 2014.

[19] W. Ding, Z. Yan, and R. H. Deng, "Secure encrypted data deduplication with ownership proof and user revocation," in *Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing*, Springer, Cham, Switherland, 2017.

[20] J. Xu, E.-C. Chang, and J. Zhou, "Weak leakage-resilient client-side deduplication of encrypted data in cloud storage," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, ASIA CCS '13*, May 2013.

[21] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks (SecureComm '08)*, ACM, September 2008.

[22] C. C. Erway et al., "Dynamic provable data possession," *ACM Transactions on Information and System Security*, vol. 17, no. 4, article 15, 2015.

[23] H. Wang, "Proxy provable data possession in public clouds," *IEEE Transactions on Services Computing*, vol. 6, no. 4, pp. 551–559, 2013.

[24] Y. Zhu, H. Hu, G.-J. Ahn, and M. Yu, "Cooperative provable data possession for integrity verification in multicloud storage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 12, pp. 2231–2244, 2012.

[25] X. Liu et al., "One-tag checker: Message-locked integrity auditing on encrypted cloud deduplication storage," in *Proceedings of the IEEE Conference on Computer Communications, INFOCOM '17*, IEEE, May 2017.

[26] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, Springer, Berlin, Germany, 2008.

[27] Q. Wang et al., "Enabling public verifiability and data dynamics for storage security in cloud computing," in *Proceedings of the European Symposium on Research in Computer Security*, Springer, Berlin, Germany, 2009.

[28] J. Xu and E.-C. Chang, "Towards efficient proofs of retrievability," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2012*, pp. 79-80, Seoul, Republic of Korea, May 2012.

[29] J. Li, X. Tan, X. Chen, D. S. Wong, and F. Xhafa, "OPoR: Enabling proof of retrievability in cloud computing with resource-constrained devices," *IEEE Transactions on Cloud Computing*, vol. 3, no. 2, pp. 195–205, 2015.

[30] Z. Ren et al., *Dynamic Proofs of Retrievability for Coded Cloud Storage Systems*, 2015.

[31] Y. Shin, J. Hur, and K. Kim, "Security weakness in the Proof of Storage with Deduplication," *IACR Cryptology ePrint Archive*, vol. 2012, p. 5, 2012.

[32] T. Youn, K. Chang, K. Rhee, and S. U. Shin, "Efficient client-side deduplication of encrypted data with public auditing in cloud storage," *IEEE Access*, vol. 6, pp. 26578–26587, 2018.