Anti-Leakage Client-Side Deduplication with Ownership Management in Fog Computing

Hua Ma, Guo-Hua Tian, and Lin-Chao Zhang
(Corresponding author: Guo-Hua Tian)

School of Mathematics and Statistics, Xidian University
Xi'an 710126, China
(Email: gh_tian0621@163.com)

(Received May 16, 2018; Revised and Accepted Oct. 18, 2018; First Online Sept. 21, 2019)

Abstract

In commercial fog computing, block-level client-side deduplication (BC-Dedu) can be used to save storage space and network bandwidth. However, the existing BC-Dedu schemes cannot support ownership management, which leads to the degradation of forward and backward secrecy of the outsourced data. Besides, BC-Dedu schemes are vulnerable to the side information leakage issue since the existence of data is revealed to the outside adversary. In this paper, we propose an anti-leakage BC-Dedu scheme that supports ownership management in fog computing. Specifically, we present a dual-level ownership list and key update mechanism to achieve ownership management in the proposed scheme. Besides, we construct a novel deduplication protocol to alleviate the side information leakage issue. Furthermore, a dynamic data storage strategy is proposed to reduce service costs and latency. Security and performance analyses demonstrate that the proposed scheme achieves the desired security requirements while saving resource efficiently.

Keywords: Deduplication; Dynamic Data Storage; Fog Computing; Ownership Management

1 Introduction

The ever-increasing volume of users in cloud computing results in an indisputable predicament that the computing power and storage capacity of the centralized cloud will be unable to provide satisfactory services for users timely in the near future [22]. To overcome this problem, fog computing is presented by Cisco in 2012 [2], which is a hierarchical service structure consists of the central cloud, fog devices, and end-users. The fog device connects to the central cloud and other fog devices through *inter-network*, and connects to end users through *intra-network* [16]. In this architecture, the central cloud offers a wide range of low-latency computing services by using fog devices adjacent to the users. Even so, the fog computing still faces the challenge of insufficient storage resources and network bandwidth caused by the growing volume of the outsourced data. Thus, researchers try to adopt cross-user deduplication technique in fog computing to save storage space and network bandwidth [10, 11], where each data is stored only once, and the subsequent versions are deleted. All of the legitimate users access the outsourced data through a link. Besides, the central cloud maintains the deduplicated data after fog devices implement deduplication over the outsourced data. This work model provides a rapid data deduplication while reducing the pressure on the central cloud.

In commercial fog computing, the service providers are most concerned about maximizing profits while ensuring system security. A feasible method is to reduce the service costs as much as possible. Considering space savings, the block-level deduplication performs more excellent in space savings than file-level one since the identical blocks of the different file can also be deduplicated in block-level deduplication. As regards bandwidth consumption, the client-side deduplication performs more excellent than the server-side one in bandwidth savings since only the unduplicated data is required to upload to the cloud server in client-side deduplication. The BC-Dedu is no doubt the best choice for service providers. However, there are some security issues remain to be solved [18].

Primarily, the data encryption key is rarely updated after its generation [15] and the data users may remove their data from fog storage to reduce service expenses. In this case, the more frequent the data ownership change is, the greater the impact on the key information disclosure. To alleviate this issue, the deduplication scheme should prevent revoked users from accessing the plaintext of outsourced data (forward secrecy). Likewise, the unauthorized user should be deterred to access the plaintext of outsourced data before she/he obtains the valid ownership (backward secrecy) [7]. Unfortunately, most of the existing block-level deduplication schemes [3, 23] do not consider the ownership management, and the existing ownership management techniques are only suitable for file-level deduplication. Therefore, it is a significant research to realize the ownership management in block-level deduplication for better security and great space savings.

Besides, the client-side deduplication is vulnerable to the side information leakage issue, which means that the malicious adversary can learn the existence of outsourced data during the upload phase by analyzing the respond of the server, namely confirmation-of-file (CoF) attack [17]. This issue is an obstinate security flaw for file-level client-side deduplication, and most of the existing BC-Dedu schemes cannot alleviate this issue efficiently. Thus, the side information leakage issue is an opening security flaw that is worth exploring.

In this paper, we propose a BC-Dedu scheme over encrypted data in fog computing. Our main contributions are listed as follows:

- We propose a dual-level ownership list and key update mechanism to implement ownership management in the proposed BC-Dedu scheme.
- We construct a novel block-level deduplication protocol to alleviate the side information leakage issue, and this protocol also alleviates the security caused by duplicate-faking attack.
- To reduce the costs and latency of data service, we provide a dynamic data storage strategy to achieve efficient resource utilization by storing data according to service demand.

2 Related Works

Although most of the existing deduplication schemes [1, 3, 7, 9, 19, 21, 23] are presented in cloud computing rather than fog computing, the related experience is worth learning. Thus, we will introduce some representative works.

2.1 Secure Client-Side Deduplication

In the original architecture of deduplication, the cloud server deduplicates the plaintext uploaded by users [5, 6], which reveals the privacy of data users to the cloud server. For better privacy, the users encrypt their data before uploading it. Unfortunately, the general cryptographic primitives obstruct the deduplication since the different users will obtain the various ciphertext by encrypting the identical data with distinct encryption keys. To realize the deduplication over encrypted data, Douceur *et al.* [4] proposed a promising solution called Convergent Encryption (CE) that requires different users use the hash value of the data to encrypt the data. As an extension of CE, Bellare *et al.* [1] presented Message-Locked Encryption (MLE) and gave a formal privacy model PRV$-CDA and strict proof for CE. Recently, Chen *et al.* [3] extended MLE to block-level for secure large file deduplication and introduced the corresponding security model PRV$-CDA-B. Zhao *et al.* [23] presented a variant of block-level MLE [3] named updatable block-level MLE that supports the block-level data update.

Considering the side information leakage issue caused by CoF attack in client-side deduplication, Harnik *et al.* [6] proposed a scheme that resists CoF attack by utilizing the client-side and server-side deduplication alternatively according to a random threshold. Based on this work, Lee *et al.* [13] minimize the success probability of CoF attack through optimizing the security parameters to enhance the security of outsourced data. Koo *et al.* [11] proposed a hybrid deduplication protocol in conjunction with client-side and server-side deduplication to alleviate the side information leakage issue in fog computing.

To prevent the malicious users who never own the target data from acquiring valid ownership with a single hash value obtained by eavesdropping, Halevi *et al.* [5] proposed an interactive Proof of Ownership (PoW) protocol based on the Merkle Hash tree (MHT). Xu *et al.* [20] proposed a client-side deduplication scheme based on MHT [5]. However, this scheme is subject to file proof reply attack. Recently, Yang *et al.* [21] proposed a provable method of ownership proof of encrypted data, which can resist the file proof reply attack. Nevertheless, the encryption key is easily leaked. Li *et al.* [14] proposed a significant application of MHT-based PoW in deduplication and auditing scenario, which resists the file proof reply attack and realizes efficient verification by allowing multiple data blocks to be challenged simultaneously. Even so, the malicious user who owns the data may launch a duplicate-faking attack (DFA) by identifying the data and uploads a poison version in the initial upload phase. Then, the subsequent uploader only can access the poison data after uploading the data. Kutylowski *et al.* [12] proposed a novel deduplication scheme, called TrDup, which can trace the malicious user by incorporating traceable signatures with MLE. Kim *et al.* [9] proposed a novel client-side deduplication scheme to prevent data users from losing data under duplicate-faking attack by using a double-tag interaction model, where the second tag is generated by the cloud server in the initial upload phase.

2.2 Ownership Management

Considering the forward and backward secrecy of the outsourced data, Hur *et al.* [7] proposed an ownership management technique that adopts a key-encryption keys tree to realize efficient ownership management. Based on this work, Jiang *et al.* [8] presented a lazy update strategy to reduce the frequency of update. However, the default user-space of this technique does not necessarily satisfy the actual demand, especially when the number of data owners changes dramatically. Besides, this technique will lead to large costs if it is employed in block-level deduplication. Koo *et al.* [10] proposed a novel ownership management technique in file-level deduplication, which achieves the fine-grained access control efficiently without the consideration of default user-space. Unfortunately, it only can be used in file-level deduplication since its block encryption key implies the information of the file.

3 Preliminary

In this section, we introduce some necessary preliminaries for the proposed deduplication scheme.

3.1 Discrete Logarithm Problem (DLP)

For a group $\mathbb{G}$ with prime order p and generator g, given $g^a \in \mathbb{G}$, where $a \in \mathbb{Z}_p$, there is no polynomial time algorithm can compute a with non-negligible probability.

3.2 Notations

We denote the empty string as ε, and let $[i] = \{1, \ldots, i\}$ for $i \in \mathbb{N}$. If x is a vector, we denote $|\mathrm{x}|$ as its dimension, and denote $\mathrm{x}[i]$ as the i-th component of x, and define that $\mathrm{x}[i,j] = \mathrm{x}[i] \ldots \mathrm{x}[j]$ for $1 \leq i \leq j \leq |\mathrm{x}|$. For a finite set S, $|S|$ denotes its size and $s \xleftarrow{r} S$ represents that an element s is uniformly selected in S. An operation that employs the algorithm $\mathcal{A}$ on inputs $x_1, \ldots$ randomly is denoted as $y \xleftarrow{r} \mathcal{A}(x_1, \ldots)$. We define the guessing probability $\mathrm{GP}(X)$ and min-entropy $\mathrm{H}_\infty(X)$ of a random variable X as $\max_x \Pr[X = x] = 2^{-\mathrm{H}_\infty(X)}$. Besides, for a random variable X given a random variable Y, we denote its conditional guessing probability $\mathrm{GP}(X\,|Y)$ and conditional min-entropy $\mathrm{H}_\infty(X\,|Y)$ as $\sum_y \Pr[Y = y] \cdot \max_x \Pr[X = x\,|Y = y] = 2^{-\mathrm{H}_\infty(X|Y)}$.

3.3 Unpredictable Sources

Suppose that a polynomial-time algorithm is a source $\mathcal{M}$, which takes 1^λ as input, outputs $(\mathbf{M}, Z)$, where $\mathbf{M}$ is a message vector in $\{0,1\}^*$ and $Z \in \{0,1\}^*$ denotes some auxiliary information of $\mathbf{M}$. We denote the length of vector $\mathbf{M}$ as $n(\lambda)$, which represents the number of blocks in our context. Besides, we label the i-th block of the message $\mathbf{M}$ with $\mathbf{M}[i]$ for all $i \in [1, n(\lambda)]$. Due to the block-level deduplication architecture of the proposed scheme, we suppose the sources output message vector over $\{0,1\}^B$, where B is the size of data block. Thus, $\mathbf{M}[i] \in \{0,1\}^B$ for all i. Furthermore, we require that $\mathbf{M}[i_1] \neq \mathbf{M}[i_2]$ for all distinct $i_1, i_2 \in [n(\lambda)]$ to bar against trivial adversary. A source $\mathcal{M}$ is unpredictable if $\mathrm{GP}_{\mathcal{M}} = \max_i\{\mathrm{GP}(\mathbf{M}[i]\,|Z)\}$ is negligible.

3.4 Block-Level MLE

According to Chen et al's works [3], a block-level MLE scheme is composed of the following algorithms:

- Setup: inputs the security parameter 1^λ and returns the system parameters P.
- KeyGen: Inputs the system parameters P and a file $M = M[1]\| \ldots \|M[n]$, runs the following two sub-algorithms and outputs a master key k_{mas} and block keys $\{k_i\}_{1\leq i \leq n}$, respectively.
  1) M-KeyGen: Takes P and M as input, returns the master key k_{mas}.
  2) B-KeyGen: Takes P and $M[i]$ as input, returns the block key k_i.
- Enc: Inputs P, a block $M[i]$ and corresponding block key k_i, outputs the block ciphertext $C[i]$.
- Dec: Inputs P, a block ciphertext $C[i]$ and block key k_i, outputs the block $M[i]$ or $\perp$.
- TagGen: Inputs P and a file M, runs the following sub-algorithms and outputs file tag t and block tags $\{T_i\}_{1\leq i\leq n}$ respectively.
  1) M-TagGen: Inputs P and M, outputs the file tag t.
  2) B-TagGen: Takes P and a block $M[i]$ as input, returns the block tag T_i.
- PoWPrf: Inputs the challenge $\mathcal{Q}$ and a file M, outputs a response $\mathcal{P}$
- PoWVer: Inputs the challenge $\mathcal{Q}$, the file tag t, the block tags $T_{i1\leq i\leq n}$,and the response $\mathcal{P}$, outputs True or False.

3.5 PRV\$-CDA-B Game

Based on the architecture of block-level MLE, Chen *et al.* [3] introduced a privacy model for block-level MLE scheme, called PRV\$-CDA-B, which argues that a block-level MLE scheme is secure under chosen distribution attacks if no polynomial-time adversary $\mathcal{A}$ can win the following chosen distribution attack game PRV\$-CDA-B with non-negligible advantage:

Setup: An adversary $\mathcal{A}$ sends the challenger $\mathcal{C}$ the description of an unpredictable block-source $\mathcal{M}$, and $\mathcal{C}$ generates and returns the system parameter P to $\mathcal{A}$.

Challenge: $\mathcal{C}$ selects $b \leftarrow \{0,1\}$ randomly. If $b = 0$, $\mathcal{C}$ runs $\mathcal{M}$ as $(\mathbf{M}^0, Z) \leftarrow \mathcal{M}(\lambda)$. Otherwise, $\mathcal{C}$ chooses $\mathbf{M}^1$ from $\{0,1\}^{|\mathbf{M}^0|}$ uniformly and randomly, and set $\mathbf{M} = \mathbf{M}^b$. We denote n as the number of blocks. For each $i \in [1,n]$, $\mathcal{C}$ computes: block keys $k_i \leftarrow \mathrm{B-KeyGen}(\mathbf{M}_i)$, ciphertexts $C_i \leftarrow \mathrm{B-Enc}(k_i, \mathbf{M}_i)$, and block tags: $T_i \leftarrow \mathrm{B-TagGen}(C_i)$, as well as the file tags: $t \leftarrow \mathbf{M}-\mathrm{TagGen}(\mathbf{M})$. Finally, $\mathcal{C}$ returns auxiliary information Z, tags $T = \{t, T_1, \ldots, T_n\}$, and the ciphertexts $C = \{C_1, \ldots, C_n\}$ to $\mathcal{A}$.

Output: The adversary $\mathcal{A}$ outputs his guess b' according to (C, T, Z). If $b' = b$, then $\mathcal{A}$ wins the game.

We regard $\mathcal{A}$ as a PRV\$-CDA-B adversary and define the advantage of $\mathcal{A}$ by

$$\mathrm{Adv}^{\mathcal{A},\mathcal{M}}_{\mathrm{PRV\$-CDA-B}}(\lambda) = \left|\Pr[b = b'] - \frac{1}{2}\right|.$$

Definition 1. *A block-level MLE scheme is PRV\$-CDA-B-secure if for any $\mathcal{M}$ and any PRV\$-CDA-B adversary $\mathcal{A}$, $\mathrm{Adv}^{\mathcal{A},\mathcal{M}}_{\mathrm{PRV\$-CDA-B}}$ is negligible.*

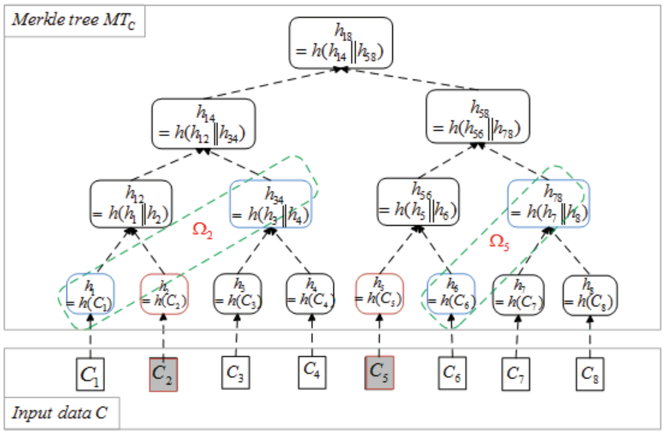


Figure 1: A Merkle Hash tree for input data with 8 blocks

3.6 Proof of Ownership

A demonstration of Li *et al.*'s [14] PoW protocol: suppose that a file C contains eight blocks. As is shown in Figure 1, the storage server constructs the MHT of C and triggers PoW process with a random challenge set $I_c = \{2, 5\}$. The prover computes h_2, h_5 of C_2, C_5 and corresponding auxiliary information $\Omega_2 = (h_1, h_{34})$, $\Omega_5 = (h_6, h_{78})$ (highlighted by green) as the ownership proof, and returns to storage server. The storage server reconstructs the root node of the MHT to verify proof:

$$h'_{18} = h(h(h(h_1 \parallel h_2) \parallel h_{34}) \parallel h(h(h_5 \parallel h_6) \parallel h_{78})).$$

If the proof is accepted, the prover is authorized to access this stored file. We employ this PoW protocol in the proposed scheme, and use the root value of MHT as the second file tag T_0.

4 System Model and Design Goals

In this section, we introduce the architecture of fog storage and define some security requirements.

4.1 Fog Storage System

The fog storage system consists of three system entities: Cloud, Fog, and End user (Figure 2).

- **Cloud**: Centralized service provider, which provides data storage and retrieval service to users, and manages the fog devices.
- **Fog**: Distributed entities, which are used as the proxy of the cloud to provide fast services.
- **End user**: Data outsourcing/retrieving entities, which are divided into initial and subsequent uploader based on whether their data has been uploaded.

We regard initial and subsequent uploader as data owners. The local fog device of a data owner is denoted as F_0, and the data storage fog device is denoted as F_s.

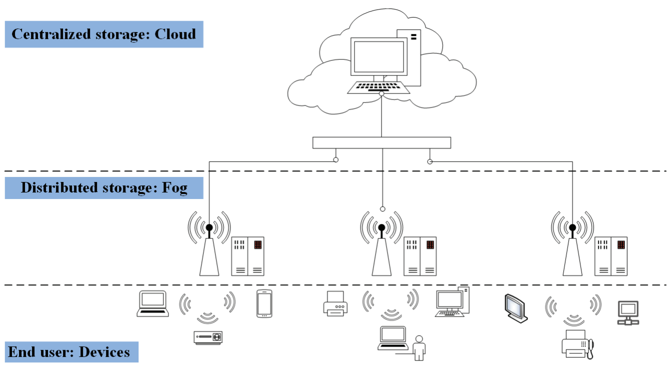


Figure 2: Fog storage system

4.2 Adversarial Model

We consider the following adversaries [8] in the proposed scheme.

- **Outside adversary**: The outside adversary may acquire some knowledge (eg., a hash value) of the file by eavesdropping, and pretend as a common user to interact with the remote server.
- **Inside adversary**: The insider adversary executes the assigned tasks honestly but would like to learn as much information of file as possible. Such as the cloud server or fog devices.

We assume that all service devices are honest-but-curious, and do not collude with outside adversary.

4.3 Security Requirements

Considering the aforementioned adversarial model, we propose the following security requirements [8]:

- **Privacy**: The proposed scheme should provide the outsourced data with PRV\$-CDA-B security [3], and prevent the plaintext of outsourced data from any illicit access.
- **Integrity**: End users should be allowed to verify the integrity of data during the data retrieval phase, and the proposed scheme should prevent users from losing data under duplicate-faking attack.
- **Leakage resilience**: Information leakage of data should be minimized as possible during data outsourcing phase.
- **Forward and backward secrecy**: In cross-user deduplication, forward secrecy means that the revoked users should be deterred to access the data stored in the remote storage. Backward secrecy means that the user should be prevented from accessing the data stored in the remote storage before she/he obtains the valid ownership.

Algorithm 1

```
Setup($1^\lambda$)
Select a hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$, and select a $\lambda$-bit prime $p$ that is the description of $H$.
Select a symmetric encryption scheme with key length $\lambda$: SKE.{KeyGen, Enc, Dec}.
Select a multiplicative group $\mathbb{G}$ with prime order $p$ such that $\mathbb{G} = \langle g \rangle$ where $g$ is the generator of $\mathbb{G}$.
Return($p$,$g$,$H$,$\mathbb{G}$)

KeyGen($M = M_1 \parallel \cdots \parallel M_n$)
for each $i \in [1,n]$
$k_i = H(M_i)$
$k_{mas} = H(M)$
Return($k_{mas}$, $\{k_i\}_{1\le i\le n}$)

Enc($\{M_i\}_{1\le i\le n}, \{k_i\}_{1\le i\le n}$)
for each $i \in [1,n]$
$C_i = \mathrm{SKE.Enc}(k_i, M_i)$
$Ck = \mathrm{SKE.Enc}(k_{mas}, k_1 \parallel \cdots \parallel k_n)$
Return($\{C_i\}_{1\le i\le n}$, $Ck$)

ReEnc($C_i$)
$r_1 \in_R \mathbb{Z}_p^*$
$Rk_i^{(1)} = g^{r_1}$
$C_i^{(1)} = C_i \cdot Rk_i^{(1)}$
Return($C_i^{(1)}$, $Rk_i^{(1)}$)

Update($C_i^{(j)}$, $Rk_i^{(j)}$)
$r_{j+1} \in_R \mathbb{Z}_p^*$
$Rk_i^{(j+1)} \leftarrow \left(Rk_i^{(j)}\right)^{r_{j+1}}$
$T^{(j)\to(j+1)} = Rk_i^{(j+1)} / Rk_i^{(j)}$
$C_i^{(j+1)} \leftarrow C_i^{(j)} \cdot T^{(j)\to(j+1)}$
Return($C_i^{(j+1)}$,$Rk_i^{(j+1)}$)

RkeyDrv($\{Rk_i^{(*)}\}_{1\le i\le n}$, $id_s$)
$C_{Rk} = id_s \cdot (Rk_1^{(*)} \parallel \cdots \parallel Rk_n^{(*)})$
Return($C_{Rk}$)

TagGen($\{C_i\}_{1\le i\le n}$, $k_{mas}$)
$T_i = H(C[i])$, for each $i \in [1,n]$
$t = g^{k_{mas}}$
Return($T_i$, $t$)

Dec($\{C_i^{(*)}\}_{1\le i\le n}$, $Ck$, $C_{Rk}$, $id_s$, $k_{mas}$)
$Rk_1^{(*)} \parallel \cdots \parallel Rk_n^{(*)} = C_{Rk}/id_s$
$k_1 \parallel \cdots \parallel k_n = \mathrm{SKE.Dec}(k_{mas}, Ck)$
for each $i \in [1,n]$
$C_i \leftarrow C_i^*/Rk_i^*$
$M_i = \mathrm{SKE.Dec}(k_i, C_i)$
Return($M = M_1 \parallel \cdots \parallel M_n$)
```

5 The Proposed Scheme

In this section, the proposed scheme is described in detail. The necessary algorithms are defined in Algorithm 1.

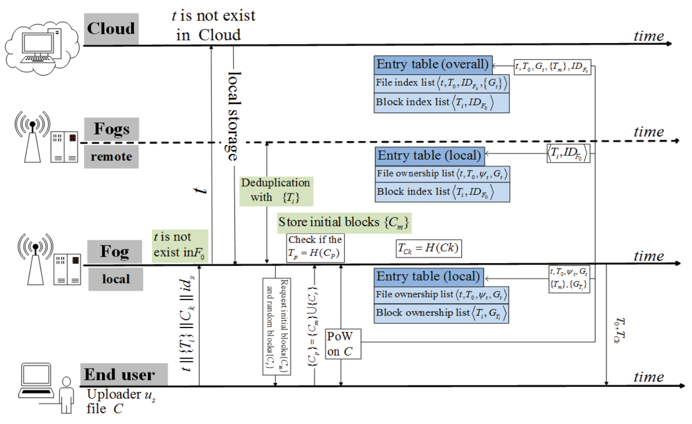
5.1 Overview

In general, our scheme first runs the file-level deduplication, and performs block-level deduplication when the data does not exist in fog computing. In the upload phase, the initial uploader is required to upload the unduplicated blocks as well as some duplicate blocks selected randomly, the subsequent uploader is requested for some random duplicate blocks. All of the random blocks will be discarded. In this way, data users cannot infer the existence of the file through the existence of the data block since they implement data outsource through the similar processes. After the data outsourcing, the outsourced data should be updated to ensure forward and backward secrecy while the ownership changes. To realize the ownership management in BC-Dedu scheme, we construct a dual-level ownership list to maintain the connections between files and updated blocks. Meanwhile, we design a corresponding update algorithm for a low-cost update operation. Moreover, we propose a dynamic data storage strategy that requires storage devices store blocks based on service demand to reduce the service costs and latency in fog computing.

5.2 Main Construction

5.2.1 System Setup

A trust initializer runs the **Setup** algorithm to obtain and publish the system parameters. The data owners are allowed to transfer data to fog storage.



Cloud
t is not exist in Cloud
time
local storage
Entry table (overall)
$t, T_0, G_t, \{T_m\}, ID_{F_0}$
File index list $\langle t, T_0, ID_{F_0}, \{G_t\}\rangle$
Block index list $\langle T_i, ID_{F_0}\rangle$
Fogs
remote
time
$\langle T_i, ID_{F_0}\rangle$
Entry table (local)
File ownership list $\langle t, T_0, \psi_t, G_t\rangle$
Block index list $\langle T_i, ID_{F_0}\rangle$
Deduplication with $\{T_i\}$
t
Store initial blocks $\{C_m\}$
t is not exist in F_0
Check if the $T_p = H(C_p)$
$T_{Ck} = H(Ck)$
Fog
local
time
t, T_0, ψ_t, G_t $\{T_m\}, \{G_{T_i}\}$
Entry table (local)
File ownership list $\langle t, T_0, \psi_t, G_t\rangle$
Block ownership list $\langle T_i, G_{T_i}\rangle$
$t \parallel \{T_i\} \parallel C_k \parallel id_s$
Request initial blocks $\{C_m\}$ and random blocks $\{C_r\}$
$\{C_p\} = \{C_m\} \bigcup \{C_r\}$
PoW on C
T_0, T_{Ck}
End user
Uploader u_s file C
time

Figure 3: Initial upload

5.2.2 Data Outsourcing and Deduplication

When a data owner u_s tries to upload file M to fog storage, u_s invokes the **KeyGen, Enc** and **TagGen** algorithm to compute the file tag t, block keys ciphertext Ck, block ciphertexts $\{C_i\}_{1\le i\le n}$ and tags $\{T_i\}_{1\le i\le n}$. Then, u_s sends the upload message $Upload \parallel t \parallel \{T_i\} \parallel Ck \parallel id_s$ to the local fog device F_0, where id_s is the identity of u_s. Notably, we regard the file data block that does not exist in the fog storage as initial block C_m, and turn the existing file block into subsequent block C_s, such that: $\{C_m\} \bigcup \{C_s\} = \{C_i\}_{1\le i\le n}$.

Initial Upload: As is illustrated in Figure 3, If t is not in F_0 and the central cloud, F_0 searches the initial blocks $\{C_m\}$ with $\{T_i\}_{1\le i\le n}$ in the block index list maintained by the cloud, and deduplicates the subsequent blocks with related storage devices. Besides, F_0 picks some duplicate blocks $\{C_r\}(\in \{C_s\})$ randomly, where the number of the random file blocks is determined by F_0 according to the security requirements. Then, F_0 requests u_s to return $\{C_m\} \bigcup \{C_r\}$. For each block C'_i returned by u_s, for each block C'_i, F_0 checks the $H(C'_i)$ with T_i, and retains the initial blocks $\{C_m\}$ if all the checks are passed. After that, F_0 triggers **PoW** protocol. If F_0 accepts u_s as a valid data owner, F_0 computes the tag of keys ciphertext $T_{Ck} = H(Ck)$. Finally, F_0 returns T_{Ck} and the root value T_0 of MHT to u_s for subsequent file retrieval.

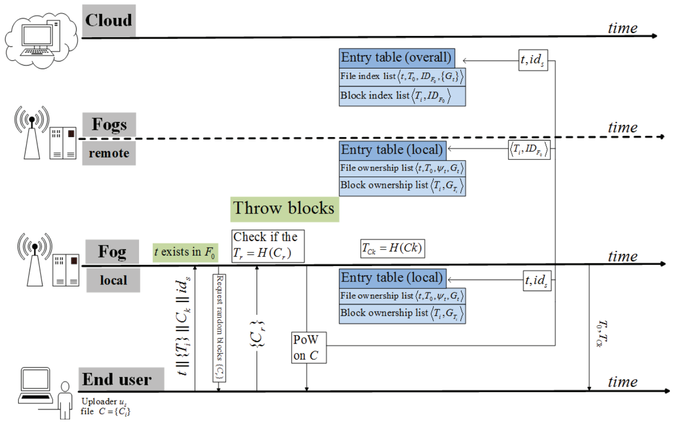

Cloud
time
Entry table (overall)
File index list $\langle t, T_0, ID_{F_0}, \{G_t\} \rangle$
Block index list $\langle T_i, ID_{F_0} \rangle$
t, id_s
Fogs
remote
time
Entry table (local)
File ownership list $\langle t, T_0, \psi_t, G_t \rangle$
Block ownership list $\langle T_i, G_{T_i} \rangle$
$\langle T_i, ID_{F_0} \rangle$
Throw blocks
Fog
local
t exists in F_0
Check if the $T_r = H(C_r)$
$T_{Ck} = H(Ck)$
time
Entry table (local)
File ownership list $\langle t, T_0, \psi_t, G_t \rangle$
Block ownership list $\langle T_i, G_{T_i} \rangle$
t, id_s
$t \parallel \{T_i\} \parallel C_k \parallel id_s$
Request random blocks $\{C_r\}$
$\{C_r\}$
PoW on C
T_0, T_{Ck}
End user
time
Uploader u_s
file $C = \{C_i\}$

Figure 4: Subsequent upload

Subsequent Upload: If t exists in F_0, the detailed process performed by F_0 is described in Figure 4. F_0 checks the consistency between t and $\{T_i\}_{1 \le i \le n}$ with ψ_t. If all the checks are passed, F_0 pretends to perform initial upload by requesting some random blocks $\{C_r\} (\in \{C_s\})$ from u_s. For each block C'_i returned by u_s, F_0 checks the $H(C'_i)$ with T_i. If all the checks are passed, F_0 discards the random blocks and triggers **PoW** protocol. If u_s's proof is accepted, F_0 computes $T_{Ck} = H(Ck)$ and performs server-side deduplication over the key ciphertext. Finally, F_0 returns T_{Ck} and T_0 to u_s.

Note that when the consistency checks between t and $\{T_i\}_{1 \le i \le n}$ are not all passed, F_0 performs the remainder procedures of initial upload, which alleviates the duplicate-faking attack by allowing the subsequent uploader to outsource the data that has been suffered the duplicate-faking attack. Besides, when t exists in another fog device F_s, F_s will employ F_0 as a proxy to perform the similar subsequent upload procedures as that executed by F_0 when t exists in F_0.

5.2.3 Onwership Management

For forward and backward secrecy, the storage devices should update the data during the following three case.

Data Upload: In this case, the data storage devices may perform the following four operation:

- For the file C uploaded by u_s initially, F_0 creates a file-level ownership list $L_F : \langle t, T_0, \psi_t, G_t \rangle$, where ψ_t is a map from t to $\{T_i\}_{1 \le i \le n}$. Then, F_0 inserts u_s into the file ownership group $G_t = \{id_s\}$.
- For an initial block C_m, F_0 creates a block-level ownership list $L_B : \langle T_m, G_{T_m} \rangle$, where the block ownership group $G_{T_m} = \{ID_F\}$ consists of the valid fog devices. Besides, F_0 runs the **ReEnc** algorithm to re-encrypts C_m, and stores the re-encrypted ciphertext and re-encryption key $\left\langle C_m^{(1)}, Rk_m^{(1)} \right\rangle$ locally. Finally, F_0 informs the central cloud to update both file-level and block-level indexes.
- For a subsequent block C_s, F_0 informs the related block storage device F_s to run **Update** algorithm to update re-encrypted block.
- For the file C uploaded by u_s subsequently, F_0 inserts u_s into G_t, and informs related block storage devices to run the **Update** algorithm to update re-encrypted blocks. Then, F_0 requests the central cloud to update the overall file-level ownership list.

Data Deletion: When $u_s (\in G_t)$ wants to delete the file C, u_s sends the file deletion message with $Delete \parallel t \parallel T_0 \parallel id_s$ to F_0. F_0 removes id_s from G_t and informs related block storage devices to run the **Update** algorithm to update re-encrypted blocks. Then, F_0 informs the central cloud to update the overall file-level ownership list.

Data Modification: We consider the following two case of the data modification.

- *Block-deletion*: when u_s wants to obtain the file C' by deleting the i-th block of C, u_s computes the t', Ck' of C' and sends $Modify \parallel t \parallel T_0 \parallel T_i \parallel t' \parallel Ck' \parallel id_s$ to F_0. Then, F_0 removes the identity id_s from G_t and informs the block storage device who stores C_i to run **Update** algorithm to update re-encrypted blocks. Subsequently, F_0 performs initial upload or subsequent upload according to whether t' exists in the system. Finally, F_0 informs the central cloud to update the overall file-level ownership list.
- Block-modification: when u_s wants to obtain the file C' by modifying the i-th block C_i of file C with C'_i. u_s computes the new t', T'_i, C'_i, Ck' , then sends the modification message $Modify \parallel t \parallel T_0 \parallel T_i \parallel t' \parallel T'_i \parallel C'_i \parallel Ck' \parallel id_s$ to F_0, Finally, F_0 performs the similar procedures as *block-deletion.*

5.2.4 Retrieval

When $u_s (\in G_t)$ wants to retrieves the file M that is stored in F_0, u_s sends data retrieval message $Retrieval \parallel$

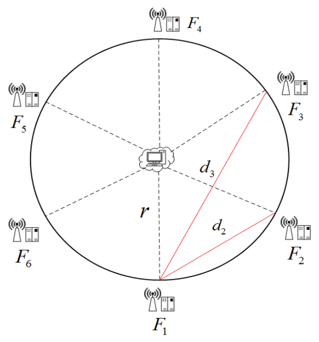


Figure 5: Ideal system model of fog storage

$t \parallel T_0 \parallel T_{Ck} \parallel id_s$ to F_0. F_0 validates the validity of the u_s through the file-level ownership list L_F, and requests all related storage devices F_s for corresponding file blocks and re-encryption key. Then, F_s validates the validity of F_0 through the block-level ownership list L_B, and returns the file blocks. F_0 constructs a file ciphertext $C^* = C_1^{(*)} \parallel \cdots \parallel C_n^{(*)}$, and computes a re-encryption key ciphertext C_{Rk} via **RkDrv** algorithm. Then, F_0 returns $\langle C^*, Ck, C_{R_k}\rangle$ to u_s. u_s takes the C^*, Ck, and C_{R_k} as input to run the **Dec** algorithm to obtain the file M.

5.2.5 Dynamic Data Storage

In fog computing, the central cloud offers a wide range of low-latency computing services by using fog device adjacent to the users, which means that the distance from local fog device to end user (*intra-network*) is much less than that from the central cloud to this fog device (*inter-network*). Therefore, the data access costs and latency is mainly caused by *inter-network* transmission. To reduce the service costs and latency, we propose a novel dynamic data storage strategy, which can periodically store data blocks in the devices of the fog computing according to the service demand.

For a concise illustration, we establish a ideal fog storage system as shown in Figure 5, where the six fog devices are distributed uniformly. Suppose that the distance from each fog device to cloud is r, the distance from j-th fog device to F_1 is d_j, and $\{d_j\}_{1\le j\le 6} = \{0, r, \sqrt{3}r, 2r, \sqrt{3}r, r\}$. For a block of size S_{C_i} that stored in F_1 periodically, we assume that the number of access from j-th fog device to C_i is num_j. Then, we introduce the following notions:

$$\begin{aligned} cost_{C_i} &= \sum\nolimits_{j=2}^{n} d_j \cdot num_j \cdot S_{C_i} \\ cost'_{C_i} &= \sum\nolimits_{j=1}^{n} num_j \cdot r \cdot S_{C_i} \\ Rate_{C_i} &= cost_{C_i}/(cost_{C_i} + cost'_{C_i}). \end{aligned}$$

Where $cost_{C_i}$ is the total access costs of block C_i stored in F_1, the $cost'_{C_i}$ is the total access costs of block C_i stored in the central cloud. Notably, when $Rate_{C_i} > 0.5$, it is cost-efficient for F_1 to move local block C_i to the central cloud. Without loss of generality, we assume that each fog device accesses C_i with an identical possibility. Then, the mean distance for fog devices to access block C_i is $(4+2\sqrt{3})r/5$. Thus, $Rate_{C_i} > 0.5$ means that:

$$rate_{F_1} = \frac{num_1}{\sum_{j=1}^{n} num_j} < \frac{2\sqrt{3}-1}{4+2\sqrt{3}} = rate_0$$

Where $rate_{F_1}$ is the percentage of the accesses launched by local device in total accesses. Obviously, the $rate_0$ is a suitable criterion for data storage devices to determine the next storage location of C_i, since $rate_0$ causes less computational overhead than $Rate_{C_i}$.

In the proposed dynamic data storage strategy, the central cloud computes the $rate_0$ as a system parameter. The fog device F_s will move a local block to the central cloud if $rate_{F_s} < rate_0$. Similarly, a block stored in cloud will be moved to fog device F_j if $rate_{F_j} > rate_0$. The decrease of data access costs is mainly caused by the decrease of mean access distance of fog devices in this dynamic data storage strategy, which also reduces the latency of data service. Besides, we allow the central cloud to adjust $rate_0$ according to service demand.

6 Security Analysis

6.1 Privacy

In the proposed scheme, each data block of the unpredictable file is re-encrypted with re-encryption key $Rk_i^{(1)}$ from C_i to $C_i^{(1)}$ in the initial upload phase. When the file ownership changes, $\langle C_i^{(j)}, Rk_i^{(j)}\rangle$ is updated with $\langle C_i^{(j+1)}, Rk_i^{(j+1)}\rangle$. The re-encryption key is retained secretly in the block storage device, and is distributed securely to a valid user u_s only when u_s retrieves the corresponding file. Therefore, the outside adversary cannot decrypt the ciphertext without the valid re-encryption key.

For a stronger security demonstration, we make a compromise to assume that the outside adversary obtains the corresponding identity and decrypts all blocks from $\{C_i^{(*)}\}$ to $\{C_i\}$. In this case, bounded by the hardness assumption of **DLP**, the outside adversary plays an identical role with the inside adversary since they cannot obtain the master key k_{mas} from $t = g^{k_{mas}}$. Besides, both the inside and the outside adversary are unable to learn any information from the ciphertext blocks $\{C_i\}$. We follow give an outline of security proof based on the existing works [1, 3, 23].

Theorem 1. *Let* SKE.{KeyGen, Enc, Dec} *be a symmetric encryption scheme with key length λ, and model $H(\cdot)$ as a random oracle. If there exists an adversary $\mathcal{B}'$ that can break the KR-security with advantage* $\mathrm{Adv}_{\mathrm{KR}}^{\mathcal{B}'}(\lambda)$, *and exists an adversary $\mathcal{D}'$ that breaks the ROR-security with advantage* $\mathrm{Adv}_{\mathrm{ROR}}^{\mathcal{D}'}(\lambda)$. *There exists a PRV\$-CDA-B adversary $\mathcal{A}$ in the proposed scheme such that:*

$$\mathrm{Adv}_{\mathrm{PRV\$-CDA-B}}^{\mathcal{A},\mathcal{M}}(\lambda) \le \mathcal{O}(qn) \cdot \mathrm{Adv}_{\mathrm{KR}}^{\mathcal{B}'}(\lambda)$$

$$+\mathrm{Adv}^{\mathcal{D}'}_{\mathrm{ROR}}(\lambda) + \frac{n^2}{2^{B+1}} + \frac{qn}{2^{\mu}}$$

where $\mathcal{M}$ is a block-source with min-entropy μ, B is the length of a block, n is the number of block messages, and q is the number of queries to $H(\cdot)$ by $\mathcal{A}$.

Proof. We introduce a sequence of PRV\$-CDA-B games to prove the privacy of our scheme by transiting the game from the world of hidden bit 0 to the world of bit 1. Meanwhile, we demonstrate that each transition is indistinguishable from the security of the underlying primitive.

Game G_0: An initial game that holds a hidden bit 0.

Game G_1: Except that a table is created by the challenger $\mathcal{C}$ to track the random oracle queries when encrypting the file M, this game is identical to G_0. On the condition that all M_i are distinct, $\mathcal{C}$ will abort the game if the result of a query X has been defined as $H(X)$ during an earlier query. Thus, $\Pr[\mathrm{G}_0^{\mathcal{A}}] \leq \Pr[\mathrm{G}_1^{\mathcal{A}}] + \Pr[\mathrm{G}_1^{\mathcal{A}}$ sets bad$]$. Note that the total number of random oracle queries is bounded by the number of ciphertexts. By union bound, we conclude that $\Pr[\mathrm{G}_1^{\mathcal{A}}$ sets bad$] = \sum_{\mathrm{i}=0}^{\mathrm{n}-1} \frac{\mathrm{i}}{2^{\mathrm{B}}} < \frac{\mathrm{n}^2}{2^{\mathrm{B}+1}}$.

Game G_2: During the challenge phase, this game is identical to G_0 until $\mathcal{A}$ makes a "bad" query of $H(X)$, and $\mathcal{C}$ aborts game due to the "bad" query. Bounded by the KR-security of the symmetric-key encryption scheme, we argue that this occurrence is negligible.

Note that the hash value of data is used as an encryption key in symmetric encryption, and the corresponding ciphertexts are sent to $\mathcal{A}$. Suppose that an adversary $\mathcal{B}$ makes such bad queries with non-negligible probability, we can break the KR-security by building an adversary $\mathcal{B}'$. $\mathcal{B}'$ just guesses hash query j^* and the encryption index i^*. Besides, $\mathcal{B}'$ plants its own key-recovery challenge c^* in the i^*-th encryption and outputs j^*-th hash query. Thus, $\Pr[\mathrm{G}_1^{\mathcal{A}}] \leq \Pr[\mathrm{G}_2^{\mathcal{A}}] + \Pr[\mathrm{G}_2^{\mathcal{A}}$ sets bad$]$. By a hybrid argument, $\Pr[\mathrm{G}_2^{\mathcal{A}}$ sets bad$] \leq \mathrm{qn}' \cdot \mathrm{Adv}^{\mathcal{B}'}_{\mathrm{KR}}(\lambda)$.

Game G_3: A further transition is to replace all encryptions of message $\mathrm{M}[i]$ or $\mathrm{M}'[j]$ with encryptions of random messages of the same length. This is possible due to ROR-security of the symmetric-key encryption scheme. Suppose that an adversary $\mathcal{D}$ who can distinguish this game from G_2, so an adversary $\mathcal{D}'$ can be built to breaks the ROR-security as follows. In ROR game, $\mathcal{D}'$ computes the ciphertext for $\mathcal{D}$ by querying its encryption oracle, then $\mathcal{D}'$ outputs the output of $\mathcal{D}$. $\Pr[\mathrm{G}_3^{\mathcal{A}}$ sets bad$] \leq \mathrm{Adv}^{\mathcal{D}'}_{\mathrm{ROR}}(\lambda)$. Thus, $\Pr[\mathrm{G}_2^{\mathcal{A}}] \leq \Pr[\mathrm{G}_3^{\mathcal{A}}] + \Pr[\mathrm{G}_3^{\mathcal{A}}$ sets bad$]$.

Game G_4: When $\mathcal{A}$ queries $H(M[i])$ for some i, $\mathcal{C}$ aborts. Recall in G_3, for the adversary $\mathcal{A}$, all the ciphertexts are independent of the true ciphertext C. So we can bound the above probability by applying the min-entropy of $\mathcal{M}$. By union bound, we have $\Pr[\mathrm{G}_4^{\mathcal{A}}$ sets bad$] = \sum_{\mathrm{i}=1}^{\mathrm{n}} \frac{\mathrm{q}}{2^{\mu}} \leq \frac{\mathrm{qn}}{2^{\mu}}$. Therefore, $\Pr[\mathrm{G}_3^{\mathcal{A}}] \leq \Pr[\mathrm{G}_4^{\mathcal{A}}] + \Pr[\mathrm{G}_4^{\mathcal{A}}$ sets bad$]$. Moreover, **Game 4** implements exactly the case where $b = 1$ such that:

$$\mathrm{Adv}^{\mathcal{A},\mathcal{M}}_{\mathrm{PRV\$-CDA-B}}(\lambda) = \Pr[\mathrm{G}_0^{\mathcal{A}}] - \Pr[\mathrm{G}_4^{\mathcal{A}}].$$

□

6.2 Integrity

When the valid data owner u_s obtains outsourced file M' during **Retrieval** phase, u_s can verify the integrity of outsourced data based on whether the equation $t = g^{H(M')}$ holds. Furthermore, the proposed scheme alleviates the duplicate-faking attack by allowing the subsequent uploader to outsource the data that has been suffered the duplicate-faking attack. In this way, the correct version is stored in fog storage.

6.3 Leakage Resilience

In the proposed scheme, the outside adversary cannot learn the existence of outsourced data by launching CoF attack, since they cannot accurately distinguish whether they are performing an initial upload or a subsequent upload according to the respond of F_0. Specifically, when a data owner u_s tries to upload file C to local fog device F_0, four cases may occur as follows:

Case (1): All blocks of C are not in fog storage;

Case (2): Partial blocks of C are not in fog storage;

Case (3): All blocks of C are in fog storage, but C is not;

Case (4): C exists in fog storage.

During the data outsourcing phase, F_0 requests data owner to upload blocks that consist of initial blocks and some random blocks, where the number of random data blocks is determined by the F_0 according to the security requirement. When the number of the requested blocks is equal to the sum of the target file, Case (2) is indistinguishable from Case (1); When the number of initial blocks is 0, Case (2), Case (3) and Case (4) are indistinguishable. Then, all blocks returned by u_s will be checked. If all of the checks are passed, the PoW protocol will be triggered. In the actual service scenario, the Case (2), Case (3) and Case (4) are occurring in data outsourcing phase frequently, where the uploader cannot learn the existence of outsourced data from the respond of F_0. Therefore, our scheme protects the outsourced data from the side information leakage.

6.4 Forward and Backward Secrecy

In the proposed scheme, when a data owner $u_s (\in G_t)$ deletes or modifies the outsourced file C, the file storage device F_s removes u_s from file-level ownership list, and informs the related block storage devices to update the data blocks and re-encryption key. Specifically, for a block $C_i^{(j)}$, a random exponent r_{j+1} is chosen to generate the new re-encryption key $Rk_i^{(j+1)}$ that will be securely stored in the block storage device. Then, the block ciphertext $C_i^{(j)}$ is re-encrypted to $C_i^{(j+1)}$ with $Rk_i^{(j+1)}/Rk_i^{(j)}$. When a valid data owner u_s requests the outsourced file, all block re-encryption keys will be integrated with the identity of u_s and returned to u_s. In this way, our scheme ensures the forward secrecy of outsourced data since the revoked users cannot decrypt the updated ciphertext without a valid re-encryption key.

When a subsequent uploader u_s uploads file to fog storage, the file storage device F_s inserts u_s into the file-level ownership list and informs the related block storage devices to update the data blocks and re-encryption keys. All of the re-encryption keys are distributed to u_s in a secure manner during the **Retrieval** phase. Thus, the unauthorized users are unable to decrypt the updated ciphertext since they have no re-encryption key before they obtain valid ownership by uploading the data. The backward secrecy of the outsourced data is guaranteed.

7 Performance Analysis

In this section, we analyze the proposed scheme and compare it with some state-of-the-art deduplication schemes in theoretical and practical aspects.

7.1 Comparisons

Table 1: Comparison of deduplication schemes

	Ownership management	**Leakage resilience**	**Dynamic storage**
BKR [1]	×	×	×
HKSK [7]	File-level	√	×
KH [10]	File-level	×	×
Ours	Block-level	√	√

Table 1 shows the comparison results of some deduplication schemes in terms of ownership management, leakage resilience, and dynamic data storage. In addition to BKR, the remainders provide ownership management for the outsourced data by updating key. Specifically, KH and HKSK achieve ownership management in file-level deduplication, and our scheme achieves ownership management in block-level deduplication, which supports more space savings in fog storage. As regards the leakage resilience in secure deduplication, HKSK can prevent data from the side information leakage based on its server-side deduplication architecture, and the BKR and KH are vulnerable to the side information leakage since the outside adversary could learn the existence of the data by launching the CoF attack. Due to our unique deduplication protocol, the proposed scheme can resist the side information leakage efficiently.

Both BKR and HKSK do not support dynamic data storage due to their architecture of single server cloud storage. KH alleviates the service pressure on the central cloud by storing the outsourced data in fog device in a period. However, all of the outsourced data will be moved to the central cloud finally. Our scheme requires the block storage devices periodically store blocks based on the service demand to reduce service costs and latency while alleviating the pressure on the cloud server.

Table 2: Notations used in theoretical analysis

Notation	Description
C_G	Bitlength of an element in $\mathbb{G}$
C_Z	Bitlength of an element in $\mathbb{Z}_p^*$
C_H	Bitlength of a hash value
C_M	Bitlength of a file M
C_k	Bitlength of a block keys ciphertext
C_{Rk}	Bitlength of a ReEnc keys ciphertext
n	Number of blocks in file M
m	Number of unduplicate blocks of M
r	Number of requested random blocks
u	Number of challenge blocks
$\mathcal{O}$	Number of data owners of file M
e	Evaluation of bilinear map
H	Evaluation of hash function
Exp	Evaluation of exponentiation
Mul	Evaluation of multiplication
Enc	Evaluation of symmetric key encryption/decryption
E_k	Evaluation of the encryption/decryption of block keys

7.2 Efficiency Analysis

We define the notations in Table 2, which are used in the following efficiency analysis in terms of computation costs, communication overheads and storage overheads.

Computation costs: Table 3 shows the computation costs of different deduplication schemes in different phases. The initial upload includes the costs of all operations for the initial uploader to outsource a new data to the remote storage. Similarly, the subsequent upload includes the costs of the subsequent uploader to regenerate the data which exists in remote storage. Verification includes the costs caused by the PoW process, Update invokes the costs for the data update, and Retrieval invokes the costs for the decryption of outsourced data on client-side.

With regard to initial upload, subsequent upload, and retrieval phases, the computation costs of our scheme are similar to HKSK and BKR, where the additional symmetric encryption operation is caused by block keys management, and it is acceptable in large file deduplication. Besides, the computation of HK is higher than other schemes, which will be illustrated with corresponding simulation experiment in the following subsection.

Only the KH and our scheme support the PoW and ownership management. As regards the verification, the computation costs of our scheme are less than KH, since the PoW protocol employed in our scheme supports multiple blocks verification simultaneously.

Table 3: Comparison of computation costs

	Initial upload	Subsequent upload	Verification	Update	Retrieval
BKR [1]	$nEnc + 2H$	$nEnc + 2H$	-	-	$nEnc$
KHSK [7]	$nEnc + 2H$	$nEnc + 2H$	-	-	$nEnc$
KH [10]	$3e + 5Exp$ $+(n+5)Mul + 1H$	$3e + Exp$ $+(n+3)Mul + 1H$	$u(\log n + 1)H$	$(\mathcal{O}+3)Exp$ $+(n+1)Mul$	$2e + Exp$ $+(n+3)Mul$
Ours	$nEnc + E_k + 2H$	$nEnc + E_k + 2H$	$\sum_{i=1}^{u} (\log n + 2 - i)H$	$nExp + 2nMul$	$nEnc + E_k$ $+nMul$

Table 4: Comparison of communication and storage overhead

	Communication overhead			Storage overhead	
	Initial upload	Subsequent upload	Retrieval	Service provider	Data owners
BKR [1]	$C_M + 3C_H$	$2C_H$	$C_M + C_H$	$C_M + 2C_H$	$2C_H$
HKSK [7]	$C_M + C_H$	$C_M + C_H$	$C_M + 2C_H$	$C_M + 2C_H$	$2C_H$
KH [10]	$C_M + C_H + 3C_G$	$C_G + u(\log n + 1)C_H$	$C_M + 4C_G + C_H$	$C_M + \lvert\mathcal{O}\rvert C_G$	$C_H + C_Z$
Ours	$\frac{m+r}{n}C_M$ $+(n+2)C_H + C_k$	$\frac{r}{n}C_M + C_k$ $+\sum_{i=1}^{u} (\log n + 2 - i)C_H$	$C_M + 2C_H$ $+C_k + C_{Rk}$	$\frac{m}{n}C_M + (n+2)C_H$ $+C_k + C_{Rk}$	$3C_H$

Considering the Update, with the increasing volume of the data owners, our scheme consumes less time than that of the KH.

Table 4 summarizes the comparison results of different schemes in terms of communication and storage overhead. The proposed scheme supports block-level client-side deduplication, and other schemes are file-level deduplication scheme, which means that our scheme saves more storage space and bandwidth.

Communication overheads: Compared with other schemes, only the partial data blocks are requested in our scheme during the initial upload phase, which saves more bandwidth than other schemes. Despite some random blocks are requested in the subsequent upload, the additional overheads can be regarded as a tradeoff since our scheme obtains a better leakage resilience than the common client-side deduplication and more bandwidth savings than the server-side deduplication.

Storage overheads: Based on the architecture of block-level deduplication, our scheme provides more efficient space savings than other schemes. Furthermore, considering the implementation of ownership management in the proposed block-level deduplication scheme, the increase in space caused by block tags, block key ciphertext, and re-encryption keys ciphertext can be seen as a trade-off, that is negligible.

7.3 Simulation

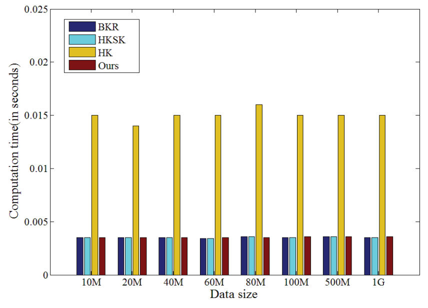
We conduct a series of simulation experiments to analyze the performance of the proposed scheme in terms of computation costs and resource utilization.

Primarily, we measure the computation costs of MLE-based schemes during different phases by using the Crypto++ library ver.5.6.2, where the SHA-256 is used as a cryptographic hash function to generate encryption key and tags, and the AES-128 with Electronic Code Book (ECB) mode is employed as an encryption/decryption function. Besides, we use the Pairing-Based Cryptography (PBC) library (Version 0.5.14) built upon the GNU Multiple-Precision (GMP) library (Version 6.0.0a) to implement HK's scheme. The size of the blocks is 1MB. All experiments are performed on a laptop with the 2.5 GHz Intel(R) Core(TM) i5-3210M CPU and 8GB memory. Note that each experimental data was obtained from the average of 20 repeated samples.

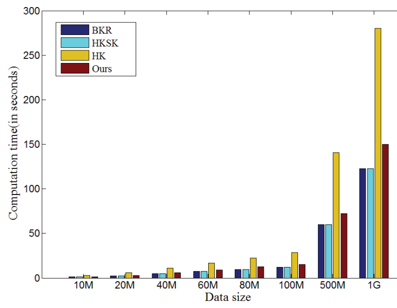
The computation costs of different schemes are shown in Figure 6. As is illustrated in Figure 6(a), the computation costs of each scheme in **Setup** phase will not change with the size of the file, and our scheme consumes almost 3.5ms in this phase, which is the same as that of BKR and HKSK. Besides, the KH's computation costs is 15ms and is higher than other schemes. During the outsourcing and retrieval phases, the computation costs of each scheme are shown in Figure 6(b) and Figure 6(c), which are proportional to the size of the outsourced data.

During the data **Outsourcing** phase, when the size of the outsourced file is 10MB, the outsourcing time is 1.2s for BKR and HKSK, 2.8s for KH, and 1.5s for our scheme. While the file size increases to 1024MB, the corresponding time is 122.9s, 280.4s and 151.2s. Compared with BKR and HKSK, the additional computation costs of our scheme is caused by the block tags generation, which is necessary for the proposed block-level deduplication.

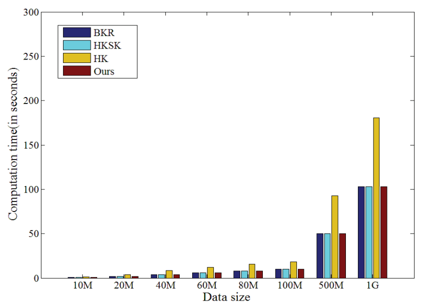
During the **Retrieval** phase, when the size of outsourced data is 10MB, the computation time is 1.02s for BKR, HKSK, 1.46s for KH, 1.1s for our scheme. While the file size increases to 1024MB, the corresponding time is 102.9s, 180.4s, 103.5s. Note that the computation time of our scheme is almost 57.4% as that of KH.



(a) Setup (Server side)

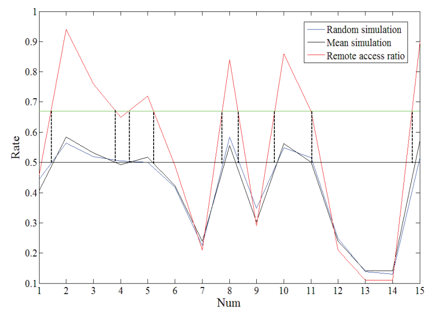


(b) Outsourcing (Client side)

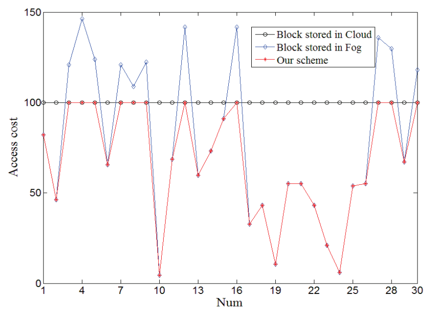


(c) Retrieval (Client side)

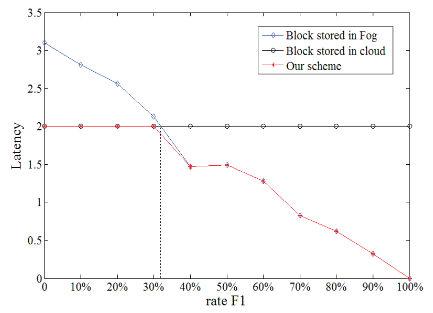
Figure 6: Comparison of computation time



(a) Comparison of access cost ratio $Rate_{C_i}$



(b) Comparison of access costs



(c) Comparison of access latency

Figure 7: Data access of inter-network

Next, we analyze the rationality of our dynamic data storage strategy under ideal fog storage model mentioned before. For simplicity, the number of total access from fogs to blocks is assumed as 100, the distance between fogs and cloud is assumed as 1 unit, the data transmission speed is 2s/unit and the size of block C_i is assumed as 1Mb. We conduct a large number of simulation experiments, and select some samples randomly for analysis.

As is depicted in Figure 7(a), a block is suitable to store in its current fog device if its $Rate_{C_i} < 0.5$. Notably, the trend of actual distance simulation (highlighted by blue) is similar to that of the mean distance simulation (highlighted by black). Besides, the value of actual distance simulation and mean distance simulation are close to 0.5 when the remote access ratio ($1 - rate_{F_1}$, highlighted by red) approaches a specific value (highlighted by green). Distinctly, the proportion $rate_{F_1}$ of the local accesses in total accesses can be used as the criterion to determine the next storage location of the block.

Under the same block access situation, we demonstrate the performance of our dynamic storage strategy in Figure 7(b) and Figure 7(c). Notably, Both the access costs and latency of our scheme are always equal to the minimum of the other two situation. Thus, our dynamic data storage strategy achieves efficient resource savings and low-latency services.

8 Conclusion

This paper has proposed a secure and efficient BC-Dedu scheme in commercial fog computing, which provides a comprehensive privacy-preservation for the outsourced data, especially in leakage resilience, forward and backward secrecy. Besides, we proposed a dynamic data storage strategy to obtain low-cost and low-latency data access services by utilizing the fog storage resource efficiently. Both the security and performance analysis demonstrate that the proposed scheme is suitable for the deduplication of large encrypted data in fog storage where ownership changes frequently.

Acknowledgment

This work is supported by the Fundamental Research Funds for the Central Universities (XJS17053, JBF181501).

References

[1] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Advances in Cryptology*, pp. 296–312, 2013.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in

ACM Edition of the Mcc Workshop on Mobile Cloud Computing, pp. 13–16, 2012.

[3] R. Chen, Y. Mu, G. Yang, and F. Guo, "Bl-mle: Block-level message-locked encryption for secure large file deduplication," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 12, pp. 2643–2652, 2015.

[4] J. R. Douceur, A. Adya, W. J. Bolosky, S. Dan, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *The 22nd International Conference on Distributed Computing Systems*, pp. 617–624, 2002.

[5] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *The 18th ACM Conference on Computer and Communications Security*, pp. 491–500, 2011.

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

[7] J. Hur, D. Koo, Y. Shin, and K. Kang, "Secure data deduplication with dynamic ownership management in cloud storage," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 11, pp. 3113–3125, 2016.

[8] S. Jiang, T. Jiang, and L. Wang, "Secure and efficient cloud data deduplication with ownership management," *IEEE Transactions on Services Computing*, pp. 1–14, 2017.

[9] K. Kim, T. Youn, N. Jho, and K. Chang, "Client-side deduplication to enhance security and reduce communication costs," *Etri Journal*, vol. 39, no. 1, pp. 116–123, 2017.

[10] D. Koo and J. Hur, "Privacy-preserving deduplication of encrypted data with dynamic ownership management in fog computing," *Future Generation Computer Systems*, vol. 78, no. 2, pp. 739–752, 2018.

[11] D. Koo, Y. Shin, J. Yun, and J. Hur, "A hybrid deduplication for secure and efficient data outsourcing in fog computing," in *IEEE International Conference on Cloud Computing Technology and Science*, pp. 285–293, 2016.

[12] M. Kutylowski, J. Li, K. Kluczniak, X. Chen, and J. Wang, "Trdup: Enhancing secure data deduplication with user traceability in cloud computing," *International Journal of Web and Grid Services*, vol. 13, no. 3, pp. 270–288, 2017.

[13] S. Lee and D. Choi, "Privacy-preserving cross-user source-based data deduplication in cloud storage," in *International Conference on ICT Convergence*, pp. 329–330, 2012.

[14] J. Li, J. Li, D. Xie, and Z. Cai, "Secure auditing and deduplicating data in cloud," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2386–2396, 2016.

[15] L. Liu, Z. Cao, and C. Mao, "A note on one outsourcing scheme for big data access control in cloud," *International Journal of Electronics and Information Engineering*, vol. 9, no. 1, pp. 29–35, 2018.

[16] M. Mukherjee, R. Matam, S. Lei, L. Maglaras, M. Ferrag, N. Choudhury, and V. Kumar, "Security and privacy in fog computing: Challenges," *IEEE Access*, vol. 5, pp. 19293–19304, 2017.

[17] P. Puzio, R. Molva, M. Onen, and S. Loureiro, "Cloudedup: Secure deduplication with encrypted data for cloud storage," in *IEEE 5th International Conference on Cloud Computing Technology and Science*, pp. 363–370, 2013.

[18] J. Singh, "Cyber-attacks in cloud computing: A case study," *International Journal of Electronics and Information Engineering*, vol. 1, no. 2, pp. 78–87, 2014.

[19] Z. Wang, Y. Lu, and G. Sun, "A policy-based deduplication mechanism for securing cloud storage," *International Journal of Electronics and Information Engineering*, vol. 2, no. 2, pp. 70–79, 2015.

[20] J. Xu, E. C.Chang, and J. Zhou, "Weak leakage-resilient client-side deduplication of encrypted data in cloud storage," in *The 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, pp. 195–206, 2013.

[21] C. Yang, J. Zhang, X. Dong, and J. Ma, "Proving method of ownership of encrypted files in cloud deduplication deletion (in chinese)," *Journal of Computer Research and Development*, vol. 52, no. 1, pp. 248–258, 2015.

[22] B. Yin, W. Shen, Y. Cheng, L .X. Cai, and Q. Li, "Distributed resource sharing in fog-assisted big data streaming," in *IEEE International Conference on Communications*, pp. 1–6, 2017.

[23] Y. Zhao and S. S. M. Chow, "Updatable block-level message-locked encryption," in *2017 ACM on Asia Conference on Computer and Communications Security*, pp. 449–460, 2017.

Biography

Hua Ma received her B.S. and M.S. degrees in Mathematics from Xidian University, China, in 1985 and 1990, respectively. She is a professor of Mathematics and statistics. Her research includes security theory and technology in electronic commerce design and analysis of fast public key cryptography theory and technology of network security.

Guohua Tian received his B.S. degree in 2016 from School of Mathematics and Information Science, Shaanxi Normal University. Now, he is a master degree student in Mathematics at Xidian University. His research focuses on network and information security.

Linchao Zhang received his B.S. degree in 2015 from College of Mathematics and Applied Mathematics, Hunan Institute of Science and Technology. Now, he is a master degree student in Mathematics at Xidian University. His research focuses on Proxy re-encryption and data security.