Achieving Efficient and Privacy-Preserving Multi-Domain Big Data Deduplication in Cloud

Xue Yang, Rongxing Lu, *Senior Member, IEEE,* Jun Shao, Xiaohu Tang, *Member, IEEE* and Ali A. Ghorbani, *Senior Member, IEEE*

Abstract—Secure data deduplication, as it can eliminate redundancies over encrypted data, has been widely developed in cloud storage to reduce storage space and communication overheads. Among them, the convergent encryption has been extensively adopted. However, it is vulnerable to brute-force attacks that can determine which plaintext in a message space corresponds to a given ciphertext. Many existing schemes have to sacrifice efficiency to resist brute-force attacks, especially for cross-domain deduplication, which is inevitably contrary to practical applications. Moreover, few existing schemes consider protecting the message equality information (i.e., whether two different ciphertexts correspond to an identical plaintext). To address the above challenges, in this paper, we propose an efficient and privacy-preserving big data deduplication scheme for a two-level multi-domain architecture. Specifically, by generating a random tag and a constant number of random ciphertexts for each data, our scheme not only ensures data confidentiality under multi-domain deduplication but also resists brute-force attacks. By allowing only the agent and cloud service provider to perform intra-deduplication and inter-deduplication, respectively, our scheme can protect the message equality information from disclosure as much as possible. Detailed security analysis shows that our scheme achieves privacy-preservation for both data content and the message equality information and data integrity while resisting brute-force attacks. Furthermore, extensive simulations demonstrate that our scheme significantly outperforms the existing competing schemes, especially the computational cost and the time complexity of the duplicate search.

Index Terms—Secure data deduplication, cross-domain deduplication, brute-force attacks, message equality information.

1 Introduction

Under big data-driven society, data deduplication technique [1] has been widely developed in cloud storage because it can significantly reduce storage costs by storing only a single copy of redundant data. Indeed, data deduplication can reduce storage costs by more than 50% in standard file systems and by more than 90% for backup applications, and these savings translate into substantial financial savings to cloud service providers and users [2]. However, considering security and privacy concerns of outsourced data, users are likely to encrypt data with their own keys before outsourcing. This impedes the cross-user deduplication since an identical data will be encrypted into different ciphertexts by different users' keys, i.e., it is challenging to identify duplicates over different ciphertexts. Thus, how to efficiently conduct data deduplication over encrypted data becomes a pressing issue.

Convergent encryption [3] provides the first viable solution to allow deduplication on encrypted data, which is formalized as the message-locked encryption later in [4]. It encrypts the data with the convergent key derived by computing the cryptographic hash value of the data itself. Since the encryption is a deterministic symmetric encryption algorithm, identical data will generate the same convergent key and ciphertext, which achieves deduplication on encrypted data. However, the convergent encryption is vulnerable to brute-force attacks due to its deterministic property [5]. To resist such attacks, some server-aided encryption schemes [5], [6], [7] have been presented. In these schemes, a domain or tenant (e.g., a company or university) deploys a dedicated key server to help affiliated users to generate the convergent key and the corresponding tag through an interactive protocol. Unfortunately, since key servers in different domains randomly select different secret keys, cross-domain deduplication is infeasible, which dramatically reduces the effectiveness of data deduplication. To support cross-domain deduplication, an inter-deduplication algorithm has recently been constructed [8]. Since only the encrypted data uploaded by the first user will be stored in the cloud, to ensure that the encrypted data can be correctly decrypted by users with ownership, the number of generated ciphertexts is linear with the number of domains. Thus, the scalability issue will inevitably arise when the number of domains explodes, which will lead to massive computational and communication overheads for users and the cloud service provider.

Moreover, the message equality information of outsourced data (i.e., the information about whether two different ciphertexts correspond to an identical plaintext) may also leak the content of encrypted data to some extent [2]. Although this information disclosure is inevitable in data deduplication, we hope to minimize such information leakage. Recently, two solutions [9], [10] have been drawn to achieve this goal. The crux of both schemes is to permit only the deduplication operator to know such information. The first scheme [9] employs an additional trusted server to complete the duplicate verification. However, the assumption that the trusted server is always online and will not be compromised is too strong to be accepted in practice [11]. The second scheme [10]

- *X. Yang and X. Tang are with the Information Security and National Computing Grid Laboratory, Southwest Jiaotong University, Chengdu, China, 610031, and X. Yang is also with the Faculty of Computer Science, University of New Brunswick, Fredericton, Canada, E3B 5A3 (e-mail: xueyang.swjtu@gmail.com, xhutang@swjtu.edu.cn).*
- *R. Lu and A. Ghorbani are with the Faculty of Computer Science, University of New Brunswick, Fredericton, Canada, E3B 5A3 (e-mail: rlu1@unb.ca, ghorbani@unb.ca).*
- *J. Shao is with the School of Computer and Information Engineering, Zhejiang Gongshang University, Zhejiang, China, 310018, and also with the Faculty of Computer Science, University of New Brunswick, Fredericton, Canada, E3B 5A3 (e-mail: chn.junshao@gmail.com).*

Corresponding author: R. Lu (e-mail: rlu1@unb.ca).

seems a good candidate because it protects the message equality information by introducing a technical method rather than an additional trusted server. Besides, this scheme efficiently achieves cross-domain deduplication and resists brute-force attacks at the same time. Unfortunately, this scheme can only support the cross-domain deduplication for two different domains.

As described in [12], secure cross-domain (or cross-user) deduplication schemes are evaluated in terms of two factors: security properties that each scheme provides and system overheads incurred on the cloud service provider and users. However, as far as we know, existing related schemes cannot efficiently achieve the strong confidentiality of outsourced data while resisting brute-force attacks. Thus, in this paper, under the similar two-level multi-domain architecture [8], [10], we propose an efficient and privacy-preserving multi-domain big data deduplication scheme in cloud storage. The main contributions of this paper are three aspects:

- First, we propose a secure multi-domain deduplication scheme that can support data deduplication not only in the same domain (called *intra-deduplication*) but also across multiple different domains (called *inter-deduplication*). Specifically, the proposed scheme generates the random convergent key and the random tag based on the bilinear pairing technique [13] to ensure data confidentiality. The proposed scheme only needs to produce a constant number of random ciphertexts to ensure that the outsourced encrypted data can be correctly decrypted by users with ownership. Moreover, the proposed scheme achieves that only the cloud service provider can perform the inter-deduplication by comparing random inter-tags derived from the Boneh-Goh-Nissim cryptosystem [14].
- Second, to improve the time complexity of duplicate search, we construct a *deduplication decision tree* based on the B+ tree [15], which works well for the big data storage system. In particular, the length of plaintext can be used to represent the keyword in the B+ tree.
- Third, we analyze the security of the proposed scheme and demonstrate that it can achieve strong data confidentiality and data integrity while resisting brute-force attacks. Besides, extensive performance evaluations justify the efficiency of the proposed scheme in terms of computational, communication and storage overheads.

The remainder of this paper is organized as follows. We will introduce the system model, threat model and design goals in Section 2, before recalling bilinear groups of composite order, the Boneh-Goh-Nissim cryptosystem and B+ tree in Section 3. Then, we present our scheme in Section 4, followed by its security analysis and performance evaluation in Sections 5 and 6, respectively. Related work is discussed in Section 7. We conclude our work in Section 8.

2 Models and Design Goals

In this section, we formalize the system model and threat model used in this paper, and identify our design goals.

2.1 System Model

Similar to [8], [10], our system is a two-level multi-domain deduplication model, which provides the *intra-deduplication* and *inter-deduplication*. More specifically, the first level contains a number of domains $\mathcal{D} = \{D_1, D_2, \dots, D_n\}$. Each domain D_i corresponding to an organization (e.g., an enterprise or a university) contains a group of affiliated users (e.g., staff in an enterprise or faculty and students in a university) and employs an agent to complete the intra-deduplication. The second level includes a cloud service provider, which conducts the inter-deduplication. Besides, a key distribution server is introduced to set up the system. Fig. 1 illustrates the system architecture of the proposed scheme and the details of each entity are described as follows.

- Key distribution server (KDS): The KDS is responsible for generating different private keys for users from different domains and a secret for the cloud service provider to perform the inter-deduplication.
- Cloud service provider (CSP): The CSP offers storage services for users. Although the CSP seems to have abundant storage space, the corresponding costs of the management and maintenance for big data are relatively expensive. Hence, the CSP prefers to conduct the inter-deduplication over outsourced data from all domains to save costs by storing only one copy.
- Users: Users from the same domain receive the same private key from the KDS, on the contrary, users from different domains possess different private keys. Based on the received private key, users can generate a random convergent key used for encrypting the data and an intra-tag used for deduplicating.
- Agent (A_i): In order to improve the efficiency of duplication search and resist an online brute-force attack launched by malicious users, an agent A_i located between users and the CSP is hired by D_i for performing intra-deduplication and transforming a random intra-tag into a random inter-tag.

The overall workflow is roughly described as follows. When a user U from D_i wants to upload data m, U generates a random intra-tag and sends it to A_i. Then, A_i performs the intra-deduplication based on the received intra-tag. If a duplicate is found, A_i returns the corresponding feedback. Otherwise, A_i transforms the intra-tag into a random inter-tag and sends it to the CSP for inter-deduplication. Note that only when the CSP does not find a duplicate, U needs to encrypt m and upload the corresponding ciphertext.

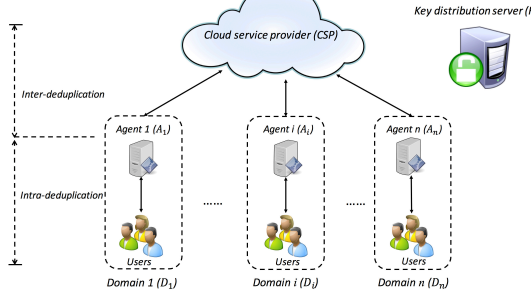


Fig. 1: System model under consideration

2.2 Threat Model

In our system, the KDS is assumed to be completely trusted and would hardly be compromised by any adversary because it only participates in system setup. Similar to the existing literature dealing with the security in cloud storage [16], [17], the CSP and A_i, considered as honest-but-curious, will honestly follow the underlying scheme, but are curious about contents of outsourced encrypted data. Specifically, based on the background knowledge about the plaintext space, the CSP would collude with A_i to guess the content of a targeted ciphertext by launching an offline brute-force attack. Furthermore, the encrypted data stored in the CSP may be altered due to physical failures [18], [19] or monetary reasons [20].

Similar to [8], users are also assumed to be honest-but-curious. Concretely, they will not frame the CSP or corresponding A_i by uploading invalid data. However, they may be interested in obtaining the content of encrypted data without ownership. Besides, if a malicious user (corrupted by the adversary) knows the plaintext space, he/she would try to launch an online brute-force attack by repeatedly executing the data upload procedure to guess the content of the targeted ciphertext. Note that users would not collude with the CSP or A_i due to their own privacy concerns and the reputation of the CSP or A_i [21].

Besides, we assume that the users from D_i connect with the corresponding agent A_i and all agencies connect with the CSP through secure channels.

2.3 Design Goals

Based on the threat model, we intend to achieve the following goals in the proposed scheme:

- *Data confidentiality*. There exist two kinds of data confidentiality in our model. One is related to the semantic security of encrypted data and tags, i.e., any adversary even corrupting the CSP and A_i or unauthorized users cannot feasibly extract any information about a plaintext from its ciphertext or tag. The other is related to the message equality information, i.e., any adversary excluding the CSP (A_i) cannot decide whether two given tags from different domains (the same domain D_i) correspond to the same data or not.
- *Data integrity*. In the design of secure data deduplication, only one copy will be stored in the CSP. Since this unique copy may be altered due to some physical failures or monetary reasons, the proposed scheme should provide data integrity to ensure that users can verify whether the downloaded data are modified or not.
- *Brute-force attack resistance*. With the background knowledge of the plaintext space, any adversary even corrupting the CSP or A_i cannot determine which plaintext corresponds to a specific tag and ciphertext through an offline brute-force attack. It is worth noting that the CSP or A_i can determine whether two ciphertexts correspond to the same plaintext by comparing received tags, but they cannot determine which plaintext in the plaintext space corresponds to these ciphertexts. Besides, the malicious user (corrupted by the adversary) tries to guess the content of the targeted ciphertext by launching online brute-force attacks. Thus, the proposed scheme should also try to resist such attacks.
- *Efficiency*. Apart from security requirements, efficiency is the most important metric for data deduplication [16]. Actually, applying secure deduplication would lead to some unavoidable costs for users and the CSP [12], e.g., tag generation, duplication search, and verification. Therefore, achieving the efficiency of computational, communication and storage overheads is also our design goal, especially in today's big data-driven society. Meanwhile, facing a vast volume of data, the time complexity of duplication search should be reduced as much as possible.

3 Preliminaries

In this section, we outline the bilinear groups of composite order [13], Boneh-Goh-Nissim cryptosystem [14] and B+ tree [15], which will serve as the basis of the proposed scheme.

3.1 Bilinear Groups of Composite Order

Given a security parameter κ, a composite bilinear parameter generator $\mathcal{G}en(\kappa)$ outputs a tuple $(p, q, \mathbb{G}, \mathbb{G}_T, e)$, where p and q are two κ-bit primes, $\mathbb{G}$ and $\mathbb{G}_T$ are two finite cyclic multiplicative groups of composite order $N = pq$, and $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a bilinear map with the following properties:

- *Bilinear*: $e(x^a, y^b) = e(x, y)^{ab}$ for all $x, y \in \mathbb{G}$, and $a, b \in \mathbb{Z}_N$.
- *Non-degeneracy*: If g is a generator of $\mathbb{G}$, then $e(g, g)$ is a generator of $\mathbb{G}_T$ with the order N.
- *Computability*: For all $x, y \in \mathbb{G}$, there exists an efficient algorithm to compute $e(x, y) \in \mathbb{G}_T$.

Some related complexity assumptions are given below. For more comprehensive descriptions, refer to [22], [23].

Definition 1. *(Computational Diffie-Hellman (CDH) Problem). The CDH problem in $\mathbb{G}$ is defined as follows: Given $g, g^a, g^b \in \mathbb{G}$ for unknown $a, b \in \mathbb{Z}_N$, compute $g^{ab} \in \mathbb{G}$.*

Definition 2. *(The n-th Bilinear Diffie-Hellman Exponent (n-BDHE) problem). The n-BDHE problem is: given a vector of $2n+1$ elements $\left(h, g, g^{\alpha}, g^{(\alpha^2)}, \ldots, g^{(\alpha^n)}, g^{(\alpha^{n+2})}, \ldots, g^{(\alpha^{2n})}\right) \in \mathbb{G}^{2n+1}$ for unknown $\alpha \in \mathbb{Z}_N$, compute $e(g, h)^{\alpha^{n+1}} \in \mathbb{G}_T$.*

It is worth noting that the input vector is missing the term $g^{\alpha^{n+1}}$ so that the bilinear map seems to be of little help in computing the required $e(g, h)^{\alpha^{n+1}}$.

Definition 3. *(Decisional n-BDHE problem). The decisional n-BDHE problem is stated as follows: given $(2n + 1)$ elements $\left(h, g, g^{\alpha}, g^{(\alpha^2)}, \ldots, g^{(\alpha^n)}, g^{(\alpha^{n+2})}, \ldots, g^{(\alpha^{2n})}\right) \in \mathbb{G}^{2n+1}$ and $R \in \mathbb{G}_T$ for an unknown random $\alpha \in \mathbb{Z}_N$, determine whether $R = e(g, h)^{\alpha^{n+1}}$ or a random element from $\mathbb{G}_T$.*

Definition 4. *(The (decisional) n-BDHE assumption) We say that the (decisional) n-BDHE assumption holds if no probabilistic and polynomial-time algorithm has advantage at least ϵ in solving the (decisional) n-BDHE problem.*

3.2 Boneh-Goh-Nissim Cryptosystem

The Boneh-Goh-Nissim cryptosystem (BGN) includes three algorithms: key generation, encryption, and decryption. The details are described as follows.

- *Key generation*: Given a security parameter κ, run $\mathcal{G}en(\kappa)$ to get a tuple $(p, q, \mathbb{G}, \mathbb{G}_T, e)$ as described in Section 3.1 and set $N = pq$. Randomly choose two generators $g, x \in \mathbb{G}$ and set $y = x^q$. The private key is p and the corresponding public key is $pk = (N, \mathbb{G}, \mathbb{G}_T, e, g, y)$.

- *Encryption*: Given a message $m \in \{0, 1, \dots, W\}$, where $W \ll q$, choose a random number $r \in \mathbb{Z}_N$, and compute the ciphertext as $C = g^m y^r \in \mathbb{G}$.
- *Decryption*: Given the ciphertext C and private key p, compute $C^p = (g^m y^r)^p = (g^p)^m$. Let $g' = g^p$, then $C^p = g'^m$. To obtain m, it suffices to compute the discrete logarithm of g'^m. Actually, when m is a short message, say $m \leqslant W$ for some small bound W, the decryption takes expected time $O(\sqrt{W})$ utilizing Pollard's lambda method [24].

3.3 B+ Tree

The B+ tree of order f is a very efficient, dynamic and balanced search tree that satisfies the following properties:

- The root node has at most f children and at least two children if it is not a leaf node.
- Each non-leaf node with n_c children contains $n_c - 1$ keywords: $(P_0, \delta_1, P_1, \delta_2, P_2, \dots, \delta_{n_c-1}, P_{n_c-1})$, where
  - $\delta_k, k = 1, 2, \dots, n_c - 1$, is a keyword with $\delta_{k-1} < \delta_k$.
  - P_k is a pointer that points to the root of the subtree. All the keywords pointed by P_{k-1} are smaller than δ_k, but greater than or equal to δ_{k-1}.
  - The number of children is satisfied $\left\lceil \frac{f}{2} \right\rceil \leqslant n_c \leqslant f$.
- Each leaf node (unless it is a root) must contain $n_c - 1$ keyword and pointer pairs: $(\delta_1, P_1, \dots, \delta_{n_c-1}, P_{n_c-1})$, where
  - $\left\lceil \frac{f}{2} \right\rceil - 1 \leqslant n_c - 1 \leqslant f - 1$.
  - All data is stored in leaf nodes, and the pointer P_k, $k = 1, \dots, n_c - 1$, points to the data that contains the keyword δ_k.
  - Each leaf node has a link to its adjacent sibling, forming the ordered linked list.
- All leaves appear in the same level.

An example of a B+ tree is illustrated in Fig. 2. Since all data are stored in leaf nodes, that is all keywords queries have the same path length (from the root to the leaf), the efficiency of search is more stable compared to other trees, e.g., the binary search tree [25] or B- tree [26]. As introduced in [15], the time complexity of the search operation in a B+ tree is $O(\log_f M)$, where M is the number of stored data. Therefore, a B+ tree can be used even when the data structure is too big to fit into main memory.

Data storage

Leaf node	keyword	data
Node 1	$\delta_1 = 1$	d_1
	$\delta_2 = 2$	d_2
	$\delta_3 = null$	$null$
Node 2	$\delta_1 = 3$	d_3
	$\delta_2 = 4$	d_4
	$\delta_3 = null$	$null$
Node 3	$\delta_1 = 5$	d_5
	$\delta_2 = 6$	d_6
	$\delta_3 = 7$	d_7

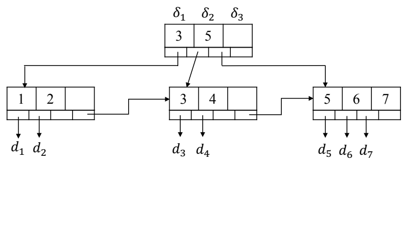


Fig. 2: A simple B+ tree of order 4 example linking keywords $1 - 7$ to data values $d_1 - d_7$.

4 The Proposed Scheme

In this section, we present our scheme, which includes three phases: system setup, data upload, and data download.

4.1 System Setup

In this phase, the trusted KDS generates private keys for users, a secret for the CSP and the corresponding system parameters. Specifically, the KDS first runs the composite bilinear parameter generator $\mathcal{G}en(\kappa_0)$ to output a tuple $(N = pq, \mathbb{G}, \mathbb{G}_T, e)$. Then, the KDS generates system parameters and private keys as follows.

- Randomly choose two generators $g, x \in \mathbb{G}$ and two numbers $\alpha, \gamma \in \mathbb{Z}_N$, and then compute $y = x^q$, $\nu = g^\gamma$ and $g_i = g^{\alpha^i}$ for $i = 1, 2, \dots, n, n+2, \dots, 2n$.
- For all users in D_i, compute the private key d_i as $d_i = g_i^\gamma$, where $i = 1, 2, \dots, n$. Note that n is the total number of domains in system and i is the identifier of D_i.
- Choose three cryptographic hash functions: $h_1 : \{0,1\}^* \to \{0,1\}^{\kappa_1}$, $h_2 : \{0,1\}^* \to \{0,1\}^{\kappa_0 - 1}$ and $h_3 : \mathbb{G} \to \{0,1\}^{\kappa_1}$, where κ_1 is the bit length of the convergent key.

Finally, the KDS sends d_i to all users in D_i and p to the CSP by secure channels, and publishes system parameters $pp = (N, \mathbb{G}, \mathbb{G}_T, e, h_1, h_2, h_3, y, g, g_1, \dots, g_n, g_{n+2}, \dots, g_{2n}, \nu)$.

In addition, based on the published system parameters, each agent A_i randomly chooses $a_i \in \mathbb{Z}_N$ as the private key, and then computes the corresponding public key g^{a_i}. Finally, A_i sends g^{a_i} to all users who belong to the domain D_i.

Note that in the practical scenario, it is important to support the dynamic addition of domains or users, especially for adding users, e.g., hiring new employees. The proposed scheme can easily support the addition of users to the existing domains. Specifically, suppose that a user U^* is added to the domain D_i, we can introduce a simple identity verification protocol that allows U^* to verify the legitimacy to the KDS. If the identity verification is passed, the KDS sends the corresponding private key d_i to U^* through the secure channel. After that, U^* can upload data in the same way as the users previously added to D_i.

For the addition of new domains, it seems impossible to increase n (i.e., the number of domains) after the scheme is once instantiated. However, similar to the concept of initializing arrays, we can initialize a relatively large n to reserve enough space to support the dynamic addition of domains to some extent. For example, if the number of domains is 10 in the current situation, then we can set n to 20. The reserved portion can be used for subsequent domain additions.

4.2 Data Upload

The data upload phase mainly includes four parts: *tag generation*, *intra-deduplication*, *inter-deduplication* and *data encryption / key recovery*. For each user U in D_i, where $i = 1, 2, \dots, n$, when U wants to upload the data m, U first generates an intra-tag for data deduplication. Then, the agent A_i performs the intra-deduplication to check the duplicate in the same domain D_i. If the duplicate does not exist, then the CSP needs to further conduct the inter-deduplication among different domains. Finally, if the duplicate is found, then U recovers the convergent key generated by the first uploader. Otherwise, U encrypts m and uploads the corresponding ciphertexts. The details are described as follows.

4.2.1 Tag generation

When a user U from D_i wants to upload data m, U chooses a random number $r_m \in \mathbb{Z}_N$, and generates a random intra-tag τ_m with the private key d_i and A_i's public key g^{a_i} as

$$\tau_m = \left(d_i^{h_2(m)} g^{a_i r_m}, g^{r_m}\right) = \left(g^{\alpha^i \cdot \gamma \cdot h_2(m) + a_i r_m}, g^{r_m}\right). \quad (1)$$

Then, U sends a message "upload$\|(L_m, \tau_m)$" to A_i. Note that L_m is the length of m, which acts as the search keyword of the deduplication decision tree (DDT) in the inter-deduplication.

4.2.2 Intra-deduplication

Upon receiving the message "upload$\|(L_m, \tau_m)$", A_i performs the intra-deduplication as follows.

1) Based on the private key a_i, compute

$$\frac{d_i^{h_2(m)} g^{a_i r_m}}{(g^{r_m})^{a_i}} = d_i^{h_2(m)} = g^{\alpha^i \cdot \gamma \cdot h_2(m)}. \quad (2)$$

Then, compute the hash of $d_i^{h_2(m)}$ as $T_m = h_3\left(d_i^{h_2(m)}\right)$.

2) Check whether a duplicated copy of m exists by comparing T_m to previously stored hash values from D_i.

- If the same hash value has already been stored, e.g., (T_{m^*}, B_{m^*}), then A_i returns "duplication$\|B_{m^*}$" to U. Note that B_{m^*} is the ciphertext used to encapsulate some information of the convergent key, which will be introduced in Section 4.2.4.
- Otherwise, A_i uses the value $d_i^{h_2(m)}$ to generate a random inter-tag $\hat{\tau}_m$ based on the BGN encryption. A_i randomly chooses $r \in \mathbb{Z}_N$, and computes

$$\hat{\tau}_m = d_i^{h_2(m)} \cdot y^r \in \mathbb{G}. \quad (3)$$

Then, A_i stores T_m in the hash table and sends the message "upload$\|(i, L_m, \hat{\tau}_m)$" to the CSP for the inter-deduplication, where i is the identifier of D_i.

At this point, the intra-deduplication part has only checked the duplicate among the outsourced data from the same domain, i.e., D_i. Thus, if the duplicate is not found in the intra-deduplication, it needs to further perform the inter-deduplication to check whether the same data has been uploaded by users from different domains.

4.2.3 Inter-deduplication

After receiving the message "upload$\|(i, L_m, \hat{\tau}_m)$" from D_i, the CSP performs the inter-deduplication to further eliminate the redundancy of data. Note that in the intra-deduplication, A_i judges whether the duplicate exists by comparing hash values. Thus, the search complexity is very efficient, i.e., $O(1)$. However, in the inter-deduplication, since inter-tags are random elements in $\mathbb{G}$ generated by BGN encryption, the same data will correspond to different inter-tags. Thus, the CSP cannot compare the hash values of inter-tags to check the duplicate. Nevertheless, if the one by one search method is adopted, the time complexity of duplicate search increases linearly with the number of encrypted data stored in the CSP, which will lead to huge search costs, especially for big data. Accordingly, we construct a deduplication search tree (DDT) based on the B+ tree for efficiently searching duplicates. The details are shown as follows.

Deduplication decision tree (DDT): According to the introduction of the B+ tree in Section 3.3, we construct a DDT of order f, as shown in Fig. 3. In the constructed DDT, the keyword used for dividing subtrees is the length of the data L_m, and each leaf node has $n_c - 1$, where $\left(\left\lceil \frac{f}{2} \right\rceil \leqslant n_c \leqslant f\right)$, pointers to point the stored data that contains the keyword L_m, i.e., $(i, L_m, \hat{\tau}_m, B_m, C_m)$. In the inter-deduplication, the CSP leverages Algorithm 1 to search a duplicate of L_m.

Algorithm 1 Search algorithm in the deduplication decision tree

```
Function: $\mathrm{Search}(L_m, \mathrm{node})$
1: The search function $\mathrm{Search}(L_m, \mathrm{node})$ is started from the
   root node, i.e., $\mathrm{node} = \mathrm{root}$, and it proceeds to a leaf node
   as follows:
2: if $\mathrm{node}$ is a leaf then
3:     return $\mathrm{node}$
4: else
5:     case 1: $L_m < \delta_1$ then
               return $\mathrm{Search}(L_m, P_0)$
       case 2: $\delta_k \leqslant L_m < \delta_{k+1}$ then
               return $\mathrm{Search}(L_m, P_k)$
       case 3: $\delta_{n_c-1} \leqslant L_m$ then
               return $\mathrm{Search}(L_m, P_{n_c-1})$
9: end if
```

Note that $1 \leqslant k < n_c - 1$, the keyword δ_k is the value of the data length L_{m_t}. Each non-leaf node contains $n_c - 1$ keywords and n_c pointers: $(P_0, \delta_1, P_1, \delta_2, P_2, \ldots, \delta_{n_c-1}, P_{n_c-1})$.

Inter-deduplication: The CSP starts checking the duplicate from the root node by comparing the data length L_m as shown in Algorithm 1. If the same value L_{m^*} is found, then for the corresponding stored data $(j, \hat{\tau}_{m^*}, B_{m^*}, C_{m^*})$, the CSP leverages Algorithm 2 to check whether the duplicate exists or not. If the data m is a duplicate, then the CSP sends "duplication$\|\mathrm{link}_{m^*}\|B_{m^*}$" to A_i, where link_{m^*} is the logical link to the data C_{m^*}. Otherwise, the CSP sends "data upload" to A_i. The details of the inter-deduplication are described in Algorithm 2, wherein lines 5 to 7, the CSP needs to check whether $i = j$ holds. The reason is that if $i = j$, it means that the received data $(i, L_m, \hat{\tau}_m)$ and stored data $(j, L_{m^*}, \hat{\tau}_{m^*}, B_{m^*}, C_{m^*})$ come from the same domain D_i. Actually, the intra-deduplication part (see Section 4.2.2) has already checked the duplicate in the same domain D_i. Only when the duplicate is not found, i.e., $m \neq m^*$, the data $(i, L_m, \hat{\tau}_m)$ will be sent to the CSP for inter-deduplication. Hence, when $i = j$, we can judge that the data m and m^* are different without the further verification. Fig. 3 shows an example of searching the duplicate.

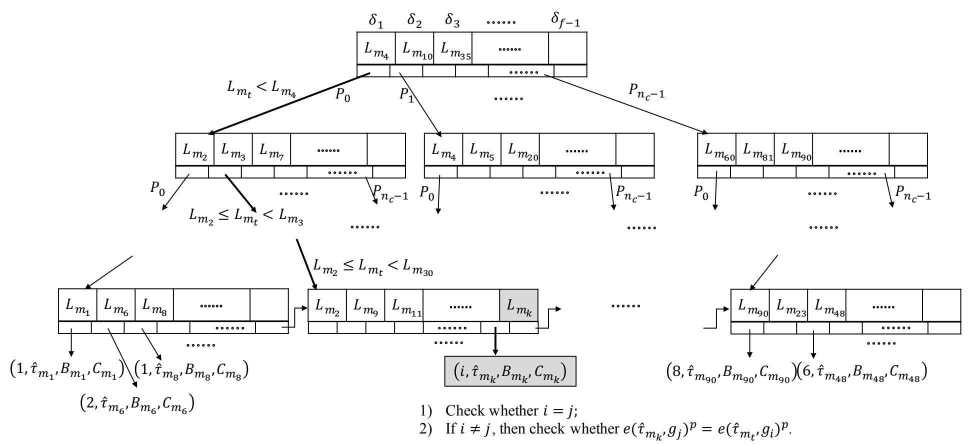
Corollary 1. *(Correctness of inter-deduplication) Suppose two inter-tags $\hat{\tau}_{m_k}$ and $\hat{\tau}_{m_t}$ are from different domains D_i and D_j, where $i, j \in \{1, \ldots, n\}$ and $i \neq j$, respectively. The equation*

$$e(\hat{\tau}_{m_k}, g_j)^p = e(\hat{\tau}_{m_t}, g_i)^p \quad (5)$$

holds, if and only if $m_k = m_t$.

Proof. Since

$$\begin{cases} \hat{\tau}_{m_k} = d_i^{h_2(m_k)} \cdot y^{r_k} = g^{\alpha^i \gamma h_2(m_k)} \cdot y^{r_k} \\ \hat{\tau}_{m_t} = d_j^{h_2(m_t)} \cdot y^{r_t} = g^{\alpha^j \gamma h_2(m_t)} \cdot y^{r_t} \end{cases}$$



δ_1 δ_2 δ_3 ...... δ_{f-1}
L_{m_4} $L_{m_{10}}$ $L_{m_{35}}$ ......
$L_{m_t} < L_{m_4}$
P_0
P_1
P_{n_c-1}
L_{m_2} L_{m_3} L_{m_7}
L_{m_4} L_{m_5} $L_{m_{20}}$
$L_{m_{60}}$ $L_{m_{81}}$ $L_{m_{90}}$
$L_{m_2} \leq L_{m_t} < L_{m_3}$
$L_{m_2} \leq L_{m_t} < L_{m_{30}}$
L_{m_1} L_{m_6} L_{m_8}
L_{m_2} L_{m_9} $L_{m_{11}}$ L_{m_k}
$L_{m_{90}}$ $L_{m_{23}}$ $L_{m_{48}}$
$(1, \hat{\tau}_{m_1}, B_{m_1}, C_{m_1})$
$(1, \hat{\tau}_{m_8}, B_{m_8}, C_{m_8})$
$(2, \hat{\tau}_{m_6}, B_{m_6}, C_{m_6})$
$(i, \hat{\tau}_{m_k}, B_{m_k}, C_{m_k})$
$(8, \hat{\tau}_{m_{90}}, B_{m_{90}}, C_{m_{90}})$
$(6, \hat{\tau}_{m_{48}}, B_{m_{48}}, C_{m_{48}})$
1) Check whether $i = j$;
2) If $i \neq j$, then check whether $e(\hat{\tau}_{m_k}, g_j)^p = e(\hat{\tau}_{m_t}, g_i)^p$.

Fig. 3: The deduplication decision tree and an example of searching the duplicate. Given the data $(j, L_{m_t}, \hat{\tau}_{m_t})$ from D_j, according to Algorithm 1, the CSP finds the same value L_{m_k} and obtains $(i, \hat{\tau}_{m_k}, B_{m_k}, C_{m_k})$. Then, the CSP executes Algorithm 2 to judge whether the duplicate exists.

Algorithm 2 Inter-deduplication algorithm

```
1: For the received data $(i, L_m, \hat{\tau}_m)$, the CSP calls the function
   $\mathrm{Search}(L_m, \mathrm{node})$ in the DDT to search whether the value
   $L_m$ has already been stored.
2: if the same value is not found then
3:    return "data upload"
4: else
5:    the same value $L_{m^*}$ is found, e.g., $(j, \hat{\tau}_{m^*}, B_{m^*}, C_{m^*})$,
      and then check whether $i = j$ holds.
6:    if $i = j$ then
7:       return "data upload"
8:    else
9:       verify whether the following equation holds
            $e(\hat{\tau}_m, g_j)^p = e(\hat{\tau}_{m^*}, g_i)^p$      (4)
10:      if Eq. (4) holds then
11:         return "duplication$\|$link$_{m^*}\|B_{m^*}$"
12:      else
13:         return "data upload"
14:      end if
15:   end if
16: end if
```

According to Corollary 1, Eq. (4) holds, if and only if $m = m_*$. That is, the duplicate can be found by verifying Eq. (4).

and $g_i = g^{\alpha^i}$, $g_j = g^{\alpha^j}$, we can compute

$$\begin{aligned} e(\hat{\tau}_{m_k}, g_j)^p &= e\big(g^{\alpha^i \gamma h_2(m_k)} \cdot y^{r_k}, g^{\alpha^j}\big)^p \\ &= e(g,g)^{\alpha^i \gamma h_2(m_k) \cdot \alpha^j \cdot p} \cdot e\big(y^{r_k}, g^{\alpha^j}\big)^p \\ &= e(g,g)^{\alpha^{i+j} \gamma p h_2(m_k)} \cdot e(x,g)^{\alpha^j r_k \cdot pq} \\ &\overset{(a)}{=} e(g,g)^{\alpha^{i+j} \gamma p h_2(m_k)} \end{aligned}$$

$$\begin{aligned} e(\hat{\tau}_{m_t}, g_i)^p &= e\big(g^{\alpha^j \gamma h_2(m_t)} \cdot y^{r_t}, g^{\alpha^i}\big)^p \\ &= e(g,g)^{\alpha^j \gamma h_2(m_t) \cdot \alpha^i \cdot p} \cdot e\big(y^{r_t}, g^{\alpha^i}\big)^p \\ &= e(g,g)^{\alpha^{i+j} \gamma p h_2(m_t)} \cdot e(x,g)^{\alpha^i r_t \cdot pq} \\ &\overset{(a)}{=} e(g,g)^{\alpha^{i+j} \gamma p h_2(m_t)} \end{aligned}$$

where (a) follows from the condition $e(x,g)^N = 1$ as the order of the group $\mathbb{G}_T$ is N.

Based on the above analyses, we can obtain that if and only if $m_k = m_t$, i.e., $h_2(m_k) = h_2(m_t)$, then

$$\begin{aligned} & e(g,g)^{\alpha^{i+j} \gamma p h_2(m_k)} = e(g,g)^{\alpha^{i+j} \gamma p h_2(m_t)} \\ \Leftrightarrow\ & e(\hat{\tau}_{m_k}, g_j)^p = e(\hat{\tau}_{m_t}, g_i)^p \end{aligned}$$

Note that $h_2 : \{0,1\}^* \to \{0,1\}^{\kappa - 1}$ is a cryptographic hash function, and the hash value satisfies $h_2(m_k) < q$ for the arbitrary data m_k. □

4.2.4 Data encryption / key recovery

Upon receiving the message "duplication$\|$link$_{m^*}\|B_{m^*}$", A_i forwards "duplication$\|B_{m^*}$" to the user U and stores $(B_{m^*}, \mathrm{link}_{m^*})$ together with T_m, i.e., $(T_m, B_{m^*}, \mathrm{link}_{m^*})$, where $m = m^*$. Otherwise, A_i directly forwards the message "data upload" to U. After receiving the feedback from A_i, if the message is "data upload", U performs data encryption operations. If the received message is "duplication$\|B_{m^*}$", U conducts key recovery operations. The details are described as follows.

Data encryption: If the message is "data upload", U encrypts the data m before outsourcing as follows.

- Randomly choose $\beta_m \in \mathbb{Z}_N$, set $K_m = e(g_{n+1}, g)^{\beta_m}$, and generate the convergent key ck_m as
$$ck_m = h_1(K_m \| m) = h_1\big(e(g_{n+1}, g)^{\beta_m} \| m\big).$$
Note that the value $e(g_{n+1}, g)$ can be computed as $e(g_i, g_{n+1-i})$.
- Generate the ciphertext of m as
$$\begin{cases} B_m = \big(g^{\beta_m}, (\nu \cdot \prod_{k \in \mathcal{I}} g_{n+1-k})^{\beta_m}\big), \\ C_m = \mathtt{SKE}_{ck_m}(m), \end{cases} \qquad (6)$$
where the set $\mathcal{I} = \{1, 2, \ldots, n\}$ is comprised of the identifiers of n domains and $\mathtt{SKE}_{ck_m}(\cdot)$ represents a fast symmetric encryption algorithm, like AES. Finally, the ciphertext tuple of the data m is (B_m, C_m).
Note that the ciphertext B_m is used to encapsulate the random value K_m. The idea of computing B_m comes

from the broadcast encryption scheme [27], which has an advantage that the number of ciphertexts is constant for any set of receivers. Therefore, we can combine the convergent encryption with this broadcast encryption scheme to guarantee that the outsourced encrypted data can be correctly decrypted by users with ownership while improving the efficiency of computation, communication, and storage. The details about the performance will be analyzed in Section 6.

After data encryption, U sends (B_m, C_m) to A_i. Upon receiving (B_m, C_m), A_i forwards it to the CSP and stores (T_m, B_m). Finally, the CSP stores $(i, L_m, \hat{\tau}_m, B_m, C_m)$ in the appropriate location of the DDT by leveraging Algorithm 1, and sends the logical link link_m to A_i for storing. That is, A_i finally stores $(T_m, B_m, \mathrm{link}_m)$.

Key recovery: If the message is "$\mathrm{duplication}\|B_{m^*}$", then U recovers the convergent key $ck_{m^*} = h_1\big(e(g_{n+1}, g)^{\beta_{m^*}}\|m^*\big)$ with the private key d_i and the data m as follows.

- Let $B_{m^*} = (B_0, B_1)$, compute
$$e(g_{n+1}, g)^{\beta_{m^*}} = \frac{e(g_i, B_1)}{e\big(d_i \cdot \prod_{\substack{k\in\mathcal{I}\\k\neq i}} g_{n+1-k+i}, B_0\big)} \quad (7)$$
- Then, U can recover ck_{m^*} with the data m as : $ck_{m^*} = h_1\big(e(g_{n+1}, g)^{\beta_{m^*}}\|m\big)$. Note that in this case, the data m is the duplicate of the data m^*, i.e., $m = m^*$.

The correctness of Eq. (7): Based on Formula (6), the ciphertext B_{m^*} can be computed as
$$B_{m^*} = (B_0, B_1) = \big(g^{\beta_{m^*}}, (\nu \cdot \prod_{k\in\mathcal{I}} g_{n+1-k})^{\beta_{m^*}}\big).$$
We can obtain
$$\begin{aligned} e(g_i, B_1) =& e\big(g_i, (\nu \cdot \prod_{k\in\mathcal{I}} g_{n+1-k})^{\beta_{m^*}}\big) \\ =& e(g_i, g_{n+1-i})^{\beta_{m^*}} \cdot e\big(g_i, \nu \cdot \prod_{\substack{k\in\mathcal{I}\\k\neq i}} g_{n+1-k}\big)^{\beta_{m^*}} \\ =& e\big(g^{\alpha^i}, g^{\alpha^{n+1-i}}\big)^{\beta_{m^*}} \cdot e\big(g, \nu^{\alpha^i} \cdot \prod_{\substack{k\in\mathcal{I}\\k\neq i}} g_{n+1-k}^{\alpha^i}\big)^{\beta_{m^*}} \\ =& e(g, g)^{\alpha^{n+1}\cdot\beta_{m^*}} \cdot e\big(g, \nu^{\alpha^i} \cdot \prod_{\substack{k\in\mathcal{I}\\k\neq i}} g_{n+1-k+i}\big)^{\beta_{m^*}} \end{aligned}$$
where $g_{n+1-k}^{\alpha^i} = \big(g^{\alpha^{n+1-k}}\big)^{\alpha^i} = g^{\alpha^{n+1-k+i}} = g_{n+1-k+i}$.
$$\begin{aligned} e\big(d_i \cdot \prod_{\substack{k\in\mathcal{I}\\k\neq i}} g_{n+1-k+i}, B_0\big) =& e\big(g_i^{\gamma} \cdot \prod_{\substack{k\in\mathcal{I}\\k\neq i}} g_{n+1-k+i}, g^{\beta_{m^*}}\big) \\ =& e\big(\nu^{\alpha^i} \cdot \prod_{\substack{k\in\mathcal{I}\\k\neq i}} g_{n+1-k+i}, g\big)^{\beta_{m^*}} \end{aligned}$$
where $g_i^{\gamma} = g^{\alpha^i\cdot\gamma} = (g^{\gamma})^{\alpha^i} = \nu^{\alpha^i}$.

Accordingly, we can obtain
$$\frac{e(g_i, B_1)}{e\big(d_i \cdot \prod_{\substack{k\in\mathcal{I}\\k\neq i}} g_{n+1-k+i}, B_0\big)} = e(g, g)^{\alpha^{n+1}\cdot\beta_{m^*}} = e(g_{n+1}, g)^{\beta_{m^*}}.$$

Finally, U deletes the data m and just stores the information $(T_m, ck_m, \mathrm{label}_m)$, where $T_m = h_3(d_i^{h_2(m)})$, label_m is the label of the data m, e.g., the name or feature used for identifying the data m). The reason for using label_m is to make it easy for users to identify each data. Note that if m is the duplicate, then $ck_m = ck_{m^*} = h_1\big(e(g_{n+1}, g)^{\beta_{m^*}}\|m\big)$, where $e(g_{n+1}, g)^{\beta_{m^*}}$ is generated by the first uploader. Otherwise, $ck_m = h_1\big(e(g_{n+1}, g)^{\beta_m}\|m\big)$ is generated by himself.

4.3 Data Download

After a period of time, when the user U from D_i wants to download an outsourced data m, U uses the corresponding label label_m to find the stored T_m, and then sends a request "$\mathrm{download}\|T_m$" to A_i. After receiving the request "$\mathrm{download}\|T_m$", A_i finds the same hash value and get the corresponding link link_m. Then, A_i returns link_m to U. After that, U can directly download the ciphertext C_m from the CSP based on the link_m.

Finally, U decrypts C_m with the stored convergent key ck_m and verifies data integrity. The details are introduced as follows.

- U recovers m' by decrypting C_m with ck_m.
- Then, with the recovered m', U computes $T_{m'} = h_3(d_i^{h_2(m')})$, and verifies the integrity by checking whether $T_{m'} = T_m$ holds. If it does not hold, it means that m has been corrupted, i.e., $m' \neq m$.

Note that in our scheme, if a user from D_i has previously tried to upload the data m. After the data deduplication, no matter whether m is successfully uploaded, A_i has a record of this data, i.e., $(T_m, B_m, \mathrm{link}_m)$. Conversely, if A_i does not store the corresponding record, then it means that no user in D_i has ever tried to upload the data m, that is all users in D_i do not have the access right to this data. Therefore, users only need to interact with A_i to obtain the corresponding logical link, which can reduce the download time to a great extent compared to the interaction with the CSP.

5 Security Analysis

In this section, we analyze the security properties of our scheme. In particular, following the design goals illustrated in Section 2.3, our analysis will focus on how our scheme can achieve data confidentiality and data integrity while resisting brute-force attacks.

5.1 Data Confidentiality

Data confidentiality of our scheme includes two aspects: (1) The semantic security of encrypted data and tags; (2) The preservation of the message equality information. The details are analyzed as follows.

5.1.1 *Semantic security of encrypted data and tags*

In our scheme, any adversary even corrupting the CSP and A_i or unauthorized users cannot feasibly extract any information about a plaintext from its ciphertext or tag.

First, we analyze that A_i and the CSP cannot obtain the content from the stored or received data. Specifically, A_i can receive the data (L_m, τ_m, B_m, C_m) from the user U in D_i. With the private key a_i, A_i can obtain $g^{\alpha^i\gamma h_2(m)}$ from the intra-tag τ_m (see Eq. (2)). If A_i wants to obtain the data m from $g^{\alpha^i\gamma h_2(m)}$, A_i first needs to overcome the discrete logarithm problem (DLP) to obtain $\alpha^i\gamma h_2(m)$, and then deals with the integer factorization to obtain $h_2(m)$. At last, A_i has to compute the one-way hash function to get m. However, the DLP, integer factorization and one-way hash function have been proved to be computationally infeasible difficult problems [28]. Hence, it is infeasible to get m from the intra-tag τ_m. If A_i wants to get m from (B_m, C_m), A_i tries to get the random element $K_m = e(g_{n+1}, g)^{\beta_m}$, which is the crux of the convergent key $ck_m = h_1(K_m \| m)$. However, based on the n-BDHE hard problem (see Definition 2), A_i cannot obtain K_m from the parameters $(g^{\beta_m}, g_1, \ldots, g_n, g_{n+2}, \ldots, g_{2n})$. Besides, based on the decisional n-BDHE problem (see Definition 3), without the private key d_i, A_i cannot recover K_m from B_m (see [27] for details). Without knowing the random K_m, the convergent ciphertext C_m is a random ciphertext generated by the symmetric encryption algorithm, which satisfies the semantic security [4], [29]. Note that the value L_m denotes the size of the data m, it is a feature that every data possesses. In general, this value would not leak any information about the data content. Therefore, A_i cannot obtain any information about the data m from the obtained data (L_m, τ_m, B_m, C_m).

The CSP stores $(i, L_m, \hat{\tau}_m, B_m, C_m)$ received from A_i. Similarly, without the private key d_i, the CSP cannot obtain the data m from the ciphertext tuple (B_m, C_m). For the inter-tag $\hat{\tau}_m = g^{\alpha^i \gamma h_2(m)} y^r$ generated by the BGN encryption, the CSP possesses the secret p and can compute $(\hat{\tau}_m)^p = (g^p)^{\alpha^i \gamma h_2(m)}$. Recall in Section 3.2, when the plaintext space is small, say $W \ll q$, it is possible to compute the discrete logarithm with Pollard's lambda method. However, the parameters α, γ and $h_2(m)$ belong to a large space. At this point, the discrete logarithm becomes a hard problem. Thus, the CSP cannot obtain any private information (i.e, the data m or the private parameters α and γ).

Then, we analyze that users lacking ownership cannot read the data content from the encrypted data. For any user U in D_i, if U tries to obtain the data that does not own by himself, U can choose a random number H such that the bit length of H is $\kappa_0 - 1$. Then, U generates an intra-tag $\tau_H = (d_i^H g^{a_i r_H}, g^{r_H})$ with the private key d_i. After that, U interacts with A_i to execute the data upload phase. If an identical hash value $T_{m^*} = h_3(d_i^{h_2(m^*)})$ happens to be stored, i.e., $H = h_2(m^*)$, then U can receive the corresponding ciphertext B_{m^*}. Accordingly, U can recover the value $K_{m^*} = e(g_{n+1}, g)^{\beta_{m^*}}$ from B_{m^*} by computing Eq. (7). However, U does not possess the data m^*, and thus U cannot generate the convergent key $ck_{m^*} = h_1(K_{m^*} \| m^*)$. In fact, the probability that a randomly selected H equals to $h_2(m^*)$ is very small, i.e., $P = \frac{1}{2^{\kappa_0 - 1}}$. Thus, when the security parameter κ_0 is large enough, it is almost impossible to obtain K_{m^*} through such attempts.

5.1.2 *Preservation of message equality information*

We prove that our scheme can protect the message equality information from disclosure to some extent. Actually, in order to achieve data deduplication, the message equality information has to be leaked. Fortunately, our scheme can minimize such disclosure. Specifically, during the intra-deduplication, before obtaining the hash value T_m to compare, Eq. (2) has to be performed, i.e., $d_i^{h_2(m)} g^{a_i r_m} / (g^{r_m})^{a_i} = d_i^{h_2(m)}$. Since only A_i possesses the private key a_i, based on the CDH problem (see Definition 1), only it can compute Eq. (2). In other words, only A_i knows the message equality information in the domain D_i. In the inter-deduplication, to verify whether two inter-tags from different domains correspond to the same data, it has to verify whether Eq. (5) holds, i.e., $e(\hat{\tau}_{m_k}, g_j)^p = e(\hat{\tau}_{m_t}, g_i)^p$. Since only the CSP possesses the secret p, only the CSP can compute Eq. (5) to complete the verification. That is, only the CSP knows the message equality information in the cross-domain and other entities cannot know whether the stored ciphertext and a given tag correspond to an identical plaintext. Therefore, compared to most related works where any entity can perform the duplicate verification to obtain the message equality information, our scheme can reduce the disclosure of such information as much as possible.

5.2 Data Integrity

After obtaining the data m', it is very necessary for U to verify data integrity to ensure the stored data m' is not altered. The reason is that deduplication techniques only keep one copy. Without the assurance of data integrity, once the outsourced data is altered, U cannot obtain the correct original data and even cannot be aware of such change.

Our scheme allows users to easily verify data integrity. Specifically, U generates the value $T_{m'} = h_3(d_i^{h_2(m')})$. Recall that after the data upload phase, U stores the hash value $T_m = h_3(d_i^{h_2(m)})$. Then U checks whether $T_{m'} = T_m$ holds. As we consider hash functions h_2 and h_3 as random oracles, if $T_{m'} \neq T_m$, then $m' \neq m$, which implies that the downloaded data m' is corrupted.

5.3 Brute-Force Attacks Resistance

In this section, we analyze that our scheme can prevent the CSP, A_i or the malicious user (corrupted by the adversary) from obtaining the content of the targeted ciphertext by launching brute-force attacks.

5.3.1 *Security against offline brute-force attacks*

First, we consider an offline brute-force attack launched by the CSP or A_i on the accessible information. Suppose the CSP knows the background knowledge of the plaintext space $\mathcal{M}$, and wishes to know which data corresponds to the specific ciphertext or tag. That is, for a stored data $(i, \hat{\tau}_m, B_m, C_m)$, where $\hat{\tau}_m = d_i^{h_2(m)} y^r$, $B_m = (g^{\beta_m}, (\nu \cdot \prod_{k \in \mathcal{I}} g_{n+1-k})^{\beta_m})$ and $C_m = \mathtt{SKE}_{ck_m}(m)$, the CSP wishes to determine which data $m_t \in \mathcal{M}$ generates them by launching offline brute-force attacks.

More specifically, with the secret key p, the CSP computes $(\hat{\tau}_m)^p$ to obtain $d_i^{p \cdot h_2(m)}$. For each data $m_t \in \mathcal{M}$, the CSP first computes the hash value $h_2(m_t)$, and then wishes to generate $d_i^{h_2(m_t)p}$ to check whether $d_i^{p \cdot h_2(m_t)} = d_i^{p \cdot h_2(m)}$ holds. If it holds, then it implies that $h_2(m) = h_2(m_t)$, i.e., $m = m_t$. Unfortunately, the CSP cannot generate the valid value $d_i^{h_2(m_t)p}$ without the private key d_i. Thus, the CSP cannot obtain the content from the inter-tag by an offline brute-force attack. Moreover, without the private key d_i, the CSP cannot obtain the value $K_m = e(g_{n+1}, g)^{\beta_m}$ from B_m or public parameters due to the (decisional) n-BDHE hard problem. Accordingly, the CSP cannot generate the valid convergent key $ck_{m_t} = h_1(K_m \| m_t)$ without K_m. That is, the CSP can neither generate a valid ciphertext C_{m_t} nor decrypt C_m to judge whether $m_t = m$ or not. Therefore, the CSP cannot determine which plaintext corresponds to the specific encrypted data and tag by launching offline brute-force attacks.

In our threat model, A_i also wants to obtain the content of the stored data by launching offline brute-force attacks. Specifically, for the obtained data (τ_m, B_m, C_m), where $\tau_m = (d_i^{h_2(m)} g^{a_i r_m}, g^{r_m})$, A_i can use the private key a_i to obtain $d_i^{h_2(m)}$ from the intra-tag τ_m. Similarly, for each data $m_t \in \mathcal{M}$,

A_i computes $h_2(m_t)$ and tries to generate $d_i^{h_2(m_t)}$ to check whether $d_i^{h_2(m_t)} = d_i^{h_2(m)}$ holds. However, it cannot generate a valid $d_i^{h_2(m_t)}$ without d_i. Similar to the CSP, A_i cannot obtain useful information from the encrypted data (B_m, C_m). Therefore, A_i also cannot guess the content of the targeted ciphertext or tag by launching offline brute-force attacks.

5.3.2 *Security against online brute-force attacks*

In addition to the CSP and A_i, a malicious user (corrupted by the adversary) from the domain D_i may launch an online brute-force attack by repeatedly executing the data upload procedure. For each data $m_t \in \mathcal{M}$, he/she generates the intra-tag τ_{m_t} and tries to upload m_t to observe whether actual uploading of m_t has occurred. The online brute-force attack repeats this procedure until the duplicate exists. That is, the malicious user can know the content corresponding to the targeted ciphertext.

Although such an attack cannot be completely prevented because the user has the private key, the actual effect of the attack can be minimized. Similar to existing works associated with brute-force attacks [5], [8], [11], we adopt the rate-limiting strategy to mitigate such attack. Specifically, we use the bounded rate-limiting approach of DupLESS [5]. In this approach, each A_i limits the total number of data upload requests a user can make during a fixed interval of time, e.g., A_i can set the maximum number of times to perform Eq. (2). By doing so, we can greatly reduce the capability of the malicious user to execute this online brute-force attack.

6 Performance Evaluation

We evaluate the performance of our scheme in terms of computational, communication and storage overheads. Moreover, we give a comparison with Shin et al's scheme [8] and Jiang et al's scheme [30]. Since these three schemes use a symmetric encryption algorithm to generate the data ciphertext, we ignore related overheads of the same part in the comparison.

6.1 Theoretical analysis

In this section, we discuss the theoretical analysis and give a comparison of three schemes. Similarly to [8] and [30], we ignore the costs of the KDS since it does not participate in data upload and download phases. Besides, compared to the exponentiation and pairing operations, the computational cost of a hash operation is very fast, which can be ignored. In what follows, we describe computational costs and communication overhead of uploading and downloading the data m, and also show the related storage costs.

6.1.1 *Computational Cost*

For the sake of simplicity, we use t_m, t_{mt}, t_e, t_{et} and t_p to represent the computational cost of a multiplication in $\mathbb{G}$, a multiplication in $\mathbb{G}_T$, an exponentiation in $\mathbb{G}$, an exponentiation in $\mathbb{G}_T$ and a pairing, respectively.

1. Computational cost of our scheme. In the phase of data upload, an intra-tag τ_m is first generated, which requires $3t_e + t_m$. Then, A_i computes Eq. (2) and T_m to check the duplicate, which costs $t_e + t_m$. If the duplicate exists, then A_i returns "$\text{duplication}\|B_{m^*}$" to the user. Otherwise, A_i computes the inter-tag $\hat{\tau}_m$, which costs $t_e + t_m$. After that, the CSP receives $(i, L_m, \hat{\tau}_m)$ and performs the inter-deduplication. Based on Algorithm 2, the CSP first searches whether the value L_m is recorded. The search complexity in DDT is $O\big(\log_f |S_{all}|\big)$, where $|S_{all}|$ denotes the number of stored data in the CSP. If the same value is found, the CSP needs to verify whether Eq. (4) holds, which costs $2t_p$. Note that $e(\hat{\tau}_m, g_j)^p = e(\hat{\tau}_m, g_j^p)$. Once the system is set up, the value g_j^p, $j = 1, \ldots, n$, can be computed offline by the CSP and will remain unchanged. Thus, the cost of g_j^p can be ignored. After finishing data deduplication, if the duplicate is not found, the user encrypts m as (B_m, C_m). In addition to the cost of the C_m, the additional costs are $2t_e + t_{et}$. Note that, B_m is computed as $B_m = \big(g^{\beta_m}, (\nu \cdot \prod_{k\in\mathcal{I}} g_{n+1-k})^{\beta_m}\big)$. Similarly, once the system is set up by the KDS, the value $(\nu \cdot \prod_{k\in\mathcal{I}} g_{n+1-k})$ is a constant. Accordingly, $(n-1)t_e$ can be ignored. However, if the duplicate is found, the user needs to recover the convergent key ck_{m^*}. The crux is to obtain the value $e(g_{n+1}, g)^{\beta_{m^*}}$ by computing Eq. (7). Likewise, for each user from D_i, $d_i \cdot \prod_{k\in\mathcal{I},k\neq i} g_{n+1-k+i}$ can be considered as a constant when the system is set up. Therefore, the corresponding costs are $t_{mt} + 2t_p$.

In the phase of data download, it contains the decryption and data verification. Since we ignore the cost of the symmetric encryption algorithm, we only discuss data verification. With the decrypted data m', the user computes $T_{m'} = h_3(d_i^{h_2(m')})$, and then verifies the integrity by checking whether $T_{m'} = T_m$ holds. Thus, the cost of data download is t_e.

2. Comparison of computational cost. In order to make a comparison, we list the computational costs of three schemes in Table 1. Note that, $|S_{all}|$ and S_{D_i} denote the number of encrypted data stored in the CSP and from the D_i, respectively. l is the bit length of short hash used in [8] and usually around $5 \sim 20$ bits.

6.1.2 *Communication Overhead*

In order to facilitate the analysis, we use $|\mathbb{G}|$ and $|\mathbb{G}_T|$ to denote the bit length of an element in $\mathbb{G}$ and $\mathbb{G}_T$, respectively. $|L|$, $|link|$ and $|i|$ denote the bit length of the data length L_m, the data link link_m and the domain identifier i, respectively.

1. Communication overhead of our scheme. When a user U in D_i wants to upload data m, U first sends "$\text{upload}\|(L_m, \tau_m)$" to A_i, which costs $|L| + 2|\mathbb{G}|$ bits. If the duplicate is found, then A_i returns "$\text{duplication}\|B_{m^*}$" with $2|\mathbb{G}|$ bits in length to U. If the duplicate is not found, then A_i sends "$\text{upload}\|(i, L_m, \hat{\tau}_m)$" with $|i| + |L| + |\mathbb{G}|$ bits in length to the CSP for the further inter-deduplication. Then the CSP returns "$\text{duplication}\|link_{m^*}\|B_{m^*}$" with $|link| + 2|\mathbb{G}|$ bits in length to A_i when the duplicate is found. Otherwise, the CSP returns "upload" to A_i. After the inter-deduplication, A_i either returns "$\text{duplication}\|B_{m^*}$" with $2|\mathbb{G}|$ bits in length or directly forwards "upload" to U. If the received message is "upload", then U sends the encrypted data (B_m, C_m) to A_i. Except for the symmetric ciphertext C_m, the additional overhead is $2|\mathbb{G}|$ bits. Then, A_i forwards (B_m, C_m) to the CSP to obtain the corresponding link link_m, which costs $|link| + 2|\mathbb{G}|$ bits to transmit. If the received message is "$\text{duplication}\|B_{m^*}$", U does not need to upload any encrypted data. Finally, when the duplicate is found in the intra-deduplication, total overheads of data upload require $|L| + 4|\mathbb{G}|$ bits. If the duplicate is found in the inter-deduplication, then total overheads are $7|\mathbb{G}| + 2|L| + |i| + |link|$ bits. Inversely, when the duplicate is not found, total overheads are $7|\mathbb{G}| + 2|L| + |i| + |link|$ bits.

For data download, U sends "$\text{download}\|T_m$" to A_i to obtain the corresponding link link_m. Then, U downloads C_m from the

TABLE 1: The comparison of computational cost for uploading or downloading one data.

			Our scheme	Shin et al's scheme [8]	Jiang et al's scheme [30]
Upload	Operation costs	Duplicate in intra-dedup.	$3t_e + 2t_m + t_{mt} + 2t_p$	$4t_m + 6t_e + 4t_p$	$-$
		Duplicate in inter-dedup.	$t_{mt} + 3t_m + 5t_e + 4t_p$	$4t_m + t_{mt} + (n+7)t_e + \left(\frac{\lvert S_{all}\rvert}{2^{l-1}} + 5\right) t_p$	$2t_e + 2t_p \cdot \log_2 \lvert S_{all}\rvert$
		Duplicate does not exist	$3t_m + 7t_e + t_{et} + 2t_p$		$4t_e + t_m + 2t_p \cdot \log_2 \lvert S_{all}\rvert$
	Search complexity	Duplicate in intra-dedup.	$O(1)$	$O\left(\log_2 \lvert S_{D_i}\rvert\right)$	$-$
		Duplicate in inter-dedup.	$O\left(\log_f \lvert S_{all}\rvert\right)$	$O\left(\frac{\lvert S_{all}\rvert}{2^l}\right)$	$O\left(\log_2 \lvert S_{all}\rvert\right)$
		Duplicate does not exist			
Data download			t_e	$t_{mt} + 3t_p$	$t_e + t_m$

TABLE 2: The comparison of communication overhead for uploading or downloading one data.

		Our scheme	Shin et al's scheme [8]	Jiang et al's scheme [30]
Upload	Duplicate in intra-dedup.	$\lvert L\rvert + 4\lvert\mathbb{G}\rvert$	$5\lvert\mathbb{G}\rvert + \lvert i\rvert$	$-$
	Duplicate in inter-dedup.	$7\lvert\mathbb{G}\rvert + 2\lvert L\rvert + \lvert i\rvert + \lvert link\rvert$	$(n+5)\lvert\mathbb{G}\rvert + \lvert\mathbb{G}_T\rvert + \lvert i\rvert + l$	$2\lvert\mathbb{G}\rvert + \lvert link\rvert$
	Duplicate does not exist			$4\lvert\mathbb{G}\rvert + \kappa_0 + \lvert link\rvert$
Data download		$\lvert link\rvert + \kappa_1$	$(n+1)\lvert\mathbb{G}\rvert + \lvert\mathbb{G}_T\rvert + \lvert i\rvert + \lvert link\rvert$	$2\lvert\mathbb{G}\rvert + \kappa_0$

TABLE 3: The comparison of storage cost.

	Our scheme	Shin et al's scheme [8]	Jiang et al's scheme [30]
User	$2\kappa_1 + \lvert label\rvert$	$2\lvert\mathbb{G}\rvert$	$2\kappa_0 + \lvert link\rvert$
The agent A_i	$2\lvert\mathbb{G}\rvert + \lvert link\rvert + \kappa_1$	$-$	$-$
The CSP	$3\lvert\mathbb{G}\rvert + \lvert L\rvert + \lvert i\rvert$	$(n+1)\lvert\mathbb{G}\rvert + \lvert\mathbb{G}_T\rvert + \lvert i\rvert + l$	$4\lvert\mathbb{G}\rvert + \kappa_0$

CSP. Recall that T_m is computed from the hash function $h_3 : \mathbb{G} \to \{0,1\}^{\kappa_1}$, where κ_1 is the bit length of the convergent key. Hence, except for the C_m, additional overheads of data download require $|link| + \kappa_1$ bits.

2. Comparison of communication overhead. Table 2 presents the communication overheads of three schemes. Note that κ_0 is the security parameter and κ_1 is the bit length of the convergent key (see Section 4.1). l is the bit length of short hash used in [8] and usually around $5 \sim 20$ bits.

6.1.3 Storage Cost

In our scheme, after data upload, the user deletes the data m and just stores $(T_m, ck_m, \text{label}_m)$, which needs $2\kappa_1 + |label|$ bits. After the intra-deduplication, if the duplicate exists, then A_i does not need to store any data. Otherwise, A_i records $(T_m, B_m, \text{link}_m)$ with $2|\mathbb{G}|+|link|+\kappa_1$ bits in length. Similarly, if the duplication exists in the inter-deduplication, the CSP would not store any data. Otherwise, the CSP stores the outsourced data $(i, L_m, \hat{\tau}_m, B_m, C_m)$. Except for the symmetric ciphertext C_m, the additional storage costs for the CSP are $3|\mathbb{G}| + |L| + |i|$ bits.

Table 3 shows the storage costs of the three schemes. Note that $|label|$ represents the bit length of the data's label.

6.2 Simulation analysis

In this section, we discuss the simulation analysis and give a comparison of three schemes. Specifically, we conduct experiments with PBC [31] and OpenSSL [32] libraries running on a 2.6 GHz-processor 2 GB-memory computing machine.

6.2.1 Simulation setup

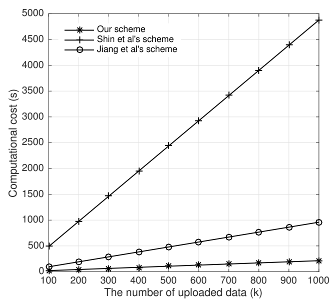
In the simulation, we instantiate the bilinear pairing with type A1, i.e., $\kappa_0 = 512$ bits. We set the bit length of the convergent key to be 256 bits, i.e., $\kappa_1 = 256$ bits. We also set $|i| = |L| = 64$ bits, $|link| = 2048$ bits and $|label| = 1024$ bits. Note that, since $|link|$ represents the bit length of the filepath, 256-bytes is sufficient to represent the usual path. Recall label_m is the label of the data m, e.g., the name or feature used for identifying the data m, thus 1024 bits are enough for the label_m. In addition, as introduced in [8], l is usually around $5 \sim 20$ bits, thus we directly set $l = 20$ bits.

For the setting of variables, suppose each user uploads or downloads k data, wherein $k\psi$ data are duplicates and $k\psi \cdot \mu$ out of $k\psi$ duplicates are found during the intra-deduplication. Note that $0 \leqslant \psi, \mu \leqslant 1$. In what follows, we analyze how the computational cost, communication overhead and storage cost change as uploaded or downloaded data number k, duplication ratio ψ, domain number n and the duplication ratio in the intra-deduplication μ.

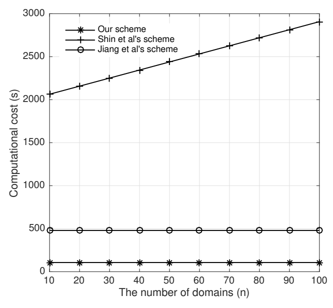
6.2.2 Simulation results

1. Computational cost. We depict the comparison of computational costs in Fig. 4. Specifically, Fig. 4(a)-4(e) show the variation of operation cost in terms of k, n, ψ, and μ. Fig. 4(f)-4(h) show the variation of the duplicate search complexity in terms of total stored data number $|S_{all}|$ and the order of the DDT f (or disjoint subset number 2^l in [8]).

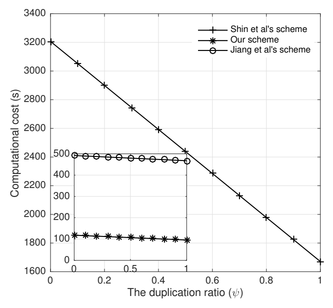
From Fig. 4, we can see that the computational efficiency of our scheme is much better than the other two schemes for all variables. The main reasons are two aspects: the efficiency of ciphertext calculations and the duplicate search complexity. On the one hand, for ensuring that the outsourced data can be correctly decrypted by users with ownership, our scheme only contains a constant number of ciphertexts, i.e., a symmetric ciphertext C_m and a key-encapsulated ciphertext B_m with two elements in $\mathbb{G}$. However, in the scheme [8], except for the symmetric ciphertext, the key-encapsulated ciphertexts



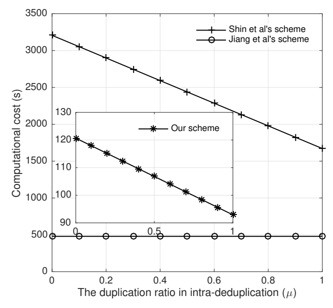
(a) Data upload.



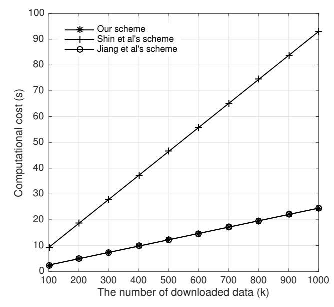
(b) Data upload.



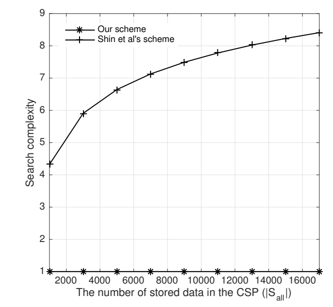
(c) Data upload.



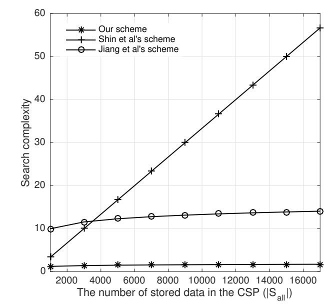
(d) Data upload.



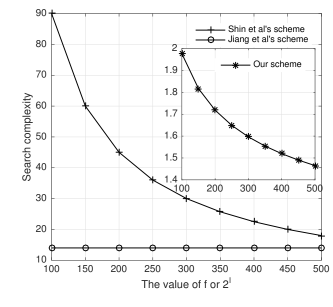
(e) Data download.



(f) Intra-deduplication.



(g) Inter-deduplication.



(h) Inter-deduplication.

Fig. 4: The comparison of computational cost.

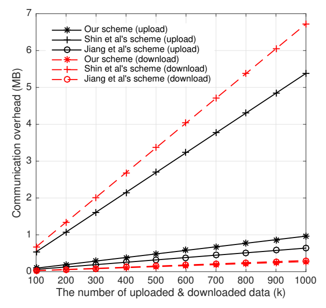
$\{e(H(m)^s, g)r, (g^{\frac{1}{y_1}})^s, \ldots, (g^{\frac{1}{y_n}})^s\}$ contain one element in $\mathbb{G}_T$ and n elements in $\mathbb{G}$, which increases linearly with n. Thus, our scheme can significantly reduce the computational costs of data encryption, as shown in Fig. 4(a)-4(d). Note that, in the scheme [30], it directly uses the hash value of the data m as the main key to obtain the symmetric key. In other words, as long as the user has $h(m)$, the symmetric key selected by the initial uploader can be obtained correctly. Thus, this scheme would not introduce a large number of additional ciphertexts as in the scheme [8]. However, it is clear that this scheme is not resistant to brute-force attacks. On the other hand, in the intra-deduplication, our scheme constructs a hash table to find the duplicate, where the search complexity is $O(1)$. But the scheme [8] uses a binary search tree to store the tag, where the search complexity is $O(\log_2 |S_{D_i}|)$. For convenience, we directly set $|S_{D_i}| = |S_{all}|/n$. Obviously, the search complexity of our scheme is more efficient than the scheme [8], as shown in Fig. 4(f). For the inter-deduplication, our scheme constructs the DDT based on the B+ tree, where the search complexity is $O\big(\log_f |S_{all}|\big)$. The scheme [30] uses the binary search tree to find the duplicate, where the search complexity is $O\left(\log_2 |S_{all}|\right)$. However, the scheme [8] introduces a short hash to divide the set S_{all} into 2^l subsets, and thus the search complexity is sub-linear, i.e., $O\big(|S_{all}|/2^l\big)$. In general, the logarithmic search complexity is more efficient than the sub-linear complexity, while the complexity of the B+ tree is more efficient than the binary search tree. Therefore, the search complexity of our scheme is more efficient than the other two, as shown in Fig. 4(g) and 4(h).

For data download, the costs of our scheme and Jiang et al's scheme [30] are less than the Shin's scheme [8] for k, as shown in Fig. 4(e). The reason is that our scheme achieves data verification by computing $T_{m^*} = h_3(d_i^{h_2(m^*)})$. However, the scheme [8] needs to verify whether $e(g, t_m) = e(H(m^*), g^{x_i})$ holds. Note that, the scheme [30] does not consider data verification. The resulting costs are related to the recovery of the symmetric key.

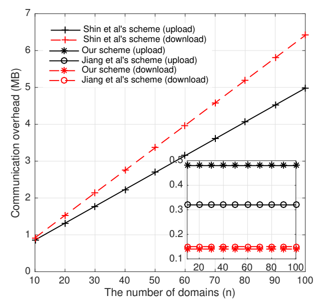
2. Communication overhead. In Fig. 5, we plot the comparison of communication overheads in terms of k, n, ψ, and μ. It is shown that our scheme significantly reduces communication overheads compared with the scheme [8]. Similar to the analysis of computational cost, the main reason is that our scheme only sends a constant number of ciphertexts to the CSP. However, for the scheme [8], the number of uploaded or downloaded ciphertexts is linear with n, which will cause more communication overheads, especially when n increases, as shown in Fig. 5(b). The communication overhead of our scheme is slightly larger than the scheme [30]. The reason is that we introduce the agent between users and the CSP to perform the intra-deduplication and forward the received messages to the CSP or users, which will increase communication overheads.

3. Storage cost. Fig. 6 shows the comparison of storage costs for these three schemes in terms of k, n, ψ, and μ. It is shown that our scheme reduces storage costs compared with the scheme [8]. The main reason is that the number of ciphertexts generated in our scheme is constant. Note that for the convenience of comparison, we summarize the storage cost of A_i to the CSP for analysis. Due to the introduction of the agent, the storage cost of our scheme is slightly larger than the scheme [30], as shown in Fig. 6(b)-6(d). But, this process helps greatly improve the efficiency of the duplicate search if the data come from the same domain, as shown in Fig. 4. More importantly, the agent can maintain a rating-limiting strategy to resist the online brute-force attacks launched by malicious users.

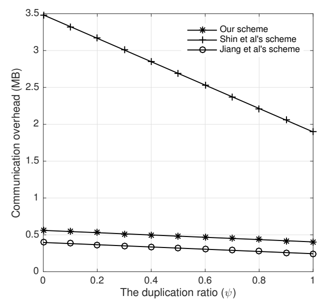
From the above analyses, our scheme is indeed efficient in terms of computational, communication and storage overheads compared with the up-to-date works, which shows the significance and practical potential of our scheme to support big data



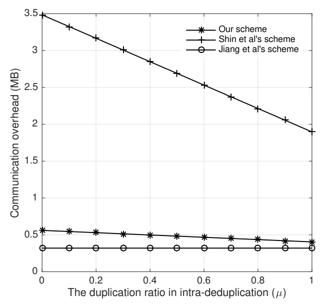
(a) Overhead of upload & download.



(b) Overhead of upload & download.

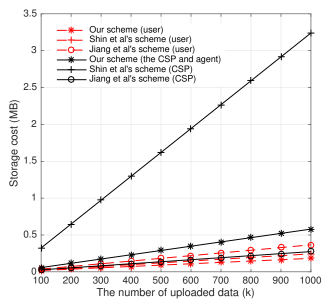


(c) Overhead of upload.

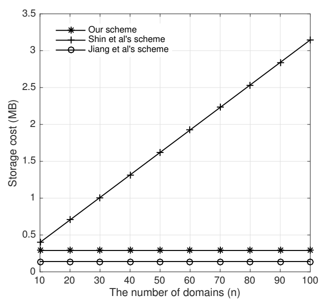


(d) Overhead of upload.

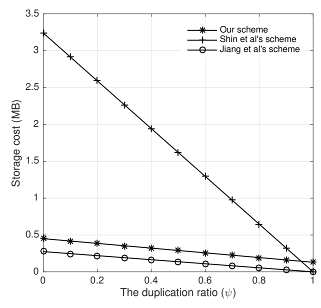
Fig. 5: The comparison of communication overhead.



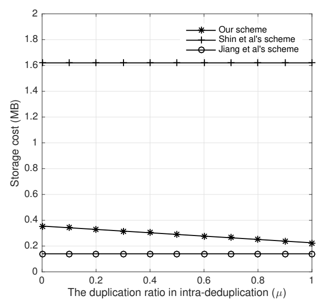
(a) Storage cost of the CSP & user.



(b) Storage cost of the CSP.



(c) Storage cost of the CSP.



(d) Storage cost of the CSP.

Fig. 6: The comparison of storage cost.

deduplication and storage.

7 Related Work

Secure data deduplication technique, as it can eliminate redundant data while achieving data confidentiality, has been widely developed by the research community [12]. The convergent encryption (CE) was the first solution that can achieve deduplication over encrypted data [3], [33]. In the CE, data is encrypted by its hash value and the corresponding tag is produced by hashing the generated ciphertext. Clearly, when different users independently encrypt the same data, they will generate the same ciphertexts and tags which can be easily deduplicated. However, the precise security guarantee of the CE was never fully proven or even stated. After that, Bellare et al. [4] formalized the CE a new cryptographic primitive called message-locked encryption (MLE) and gave a semantic security proof for unpredictable messages. Unfortunately, for predictable messages, the MLE (as well as the CE) is vulnerable to brute-force attacks due to its deterministic property [5].

In order to resist brute-force attacks, Bellare et al. [5] firstly proposed a server-aided secure deduplication scheme, called DupLESS, based on the RSA-OPRF (Oblivious Pseudo-Random Function) protocol [34]. In DupLESS, each user interacts with a centralized key server (KS) to obtain the corresponding convergent key for each uploaded data. Obviously, this centralized-KS should be always online to response key generation requests sent from users in the system. Thus, it is very vulnerable to the single point of failure, i.e., once the KS is compromised, the security of the DupLESS totally degrades into the MLE algorithm. To deal with this issue, Miao et al. [6] presented a multi-server-aided secure deduplication scheme based on a threshold blind signature [35]. In this scheme, each user interacts with at least a threshold number of KSs to generate a convergent key. Besides, Duan et al. [7] introduced a distributed key generation protocol based on the RSA threshold signature [36]. Unlike the scheme proposed in [6], this scheme employed a trusted dealer to distribute key shares for online users in the system and each user interacts with at least a threshold number of online users to acquire a convergent key. The trusted dealer plays a similar role to that of the KS in DupLESS, but it participates only in the system setup phase. In both schemes, as long as the number of compromised KSs or online users does not exceed the threshold, the security can be preserved. Unfortunately, both two schemes only can deal with the intra-deduplication, i.e., they can only remove the redundancy among outsourced data that correspond to the same KSs or users (i.e., the same domain in our system). In order to extend the deduplication into the cross-domain, Shin et al. [8] recently constructed an inter-deduplication scheme based on the Boldyreva's blind signature [37]. In this scheme, even private keys chosen by different KSs are different, the CSP can still eliminate the redundancy among outsourced data from different domains. However, since different outsourced data are encrypted by different private keys, in order to ensure these data can be correctly decrypted by users with the ownership, for each uploaded data, the number of corresponding ciphertexts is linear with the number of domains. Thus, massive computational, communication and storage overheads are incurred when the number of domains explodes.

In addition to the semantic security of encrypted data and tags, the message equality information of outsourced data (i.e., the information whether two different ciphertexts correspond to an identical plaintext) needs to be protected as much as possible. For example, Stanek et al. [9] added a trusted third party, index repository service (IRS), to complete the duplication judgment. That is, only the IRS can know the message equality information. Thus, this scheme can prevent other entities from obtaining the message equality information. However, the assumption of a trusted IRS is too strong to be accepted in practice. Furthermore, the IRS seems vulnerable because it should be always online to deal with data upload requests. Accordingly, this scheme is vulnerable to offline brute-force attacks when the IRS is compromised [11]. To improve these disadvantages, Yang et al. [10] proposed an efficient deduplication scheme that achieves data confidentiality while resisting brute-force attacks. Without requiring an additional trusted server in this scheme, any adversary excluding the CSP cannot know the message equality information, i.e., only the CSP can decide whether two given tags from different domains correspond to the same data or not. However, this scheme only can deduplicate for outsourced data from two different domains.

8 Conclusion

In this paper, we have proposed an efficient and privacy-preserving big data deduplication scheme for a two-level multi-domain architecture. Specifically, our scheme can achieve the semantic security of encrypted data while ensuring that the outsourced encrypted data can be correctly decrypted by users with ownership by generating a constant number of ciphertexts. Besides, only the CSP (the agent) can decide whether two given random inter-tags from different domains (intra-tags from the same domain) correspond to the same data or not, which minimizes the disclosure of the message equality information. Detailed security analyses demonstrate that our scheme can achieve data confidentiality and data integrity while resisting brute-force attacks. Furthermore, extensive performance evaluations show that our scheme outperforms the existing competing schemes, especially the computational cost and the time complexity of the duplicate search.

Future research includes extending the proposed scheme to achieve the ownership management and revocation because data modification or deletion operations are often requested by users. Moreover, since data deduplication techniques keep only one copy of the data, outsourced data after deduplication is vulnerable to data loss or corruption. Accordingly, future research agenda will also include extending the proposed scheme to address the reliability of outsourced data.

References

[1] D. T. Meyer and W. J. Bolosky, “A study of practical deduplication,” *TOS*, vol. 7, no. 4, pp. 14:1–14:20, 2012.

[2] 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.

[3] 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 *ICDCS*, 2002, pp. 617–624.

[4] M. Bellare, S. Keelveedhi, and T. Ristenpart, “Message-locked encryption and secure deduplication,” in *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, 2013, pp. 296–312.

[5] S. Keelveedhi, M. Bellare, and T. Ristenpart, “Dupless: Server-aided encryption for deduplicated storage,” in *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, 2013, pp. 179–194.

[6] M. Miao, J. Wang, H. Li, and X. Chen, “Secure multi-server-aided data deduplication in cloud computing,” *Pervasive and Mobile Computing*, vol. 24, pp. 129–137, 2015.

[7] Y. Duan, “Distributed key generation for encrypted deduplication: Achieving the strongest privacy,” in *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security, CCSW ’14, Scottsdale, Arizona, USA, November 7, 2014*, 2014, pp. 57–68.

[8] Y. Shin, D. Koo, J. Yun, and J. Hur, “Decentralized server-aided encryption for secure deduplication in cloud storage,” *IEEE Transactions on Services Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[9] J. Stanek and L. Kencl, “Enhanced secure thresholded data deduplication scheme for cloud storage,” *IEEE Transactions on Dependable and Secure Computing*, vol. PP, no. 99, pp. 1–1, 2016.

[10] X. Yang, R. Lu, K.-K. R. Choo, F. Yin, and X. Tang, “Achieving efficient and privacy-preserving cross-domain big data deduplication in cloud,” *IEEE Transactions on Big Data*, vol. PP, no. 99, pp. 1–1, 2017.

[11] 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, Denver, CO, USA, October 12-6, 2015*, 2015, pp. 874–885.

[12] Y. Shin, D. Koo, and J. Hur, “A survey of secure data deduplication schemes for cloud storage systems,” *ACM Comput. Surv.*, vol. 49, no. 4, pp. 74:1–74:38, 2017.

[13] D. Boneh and B. Waters, “Conjunctive, subset, and range queries on encrypted data,” in *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, 2007, pp. 535–554.

[14] D. Boneh, E. Goh, and K. Nissim, “Evaluating 2-dnf formulas on ciphertexts,” in *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, 2005, pp. 325–341.

[15] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems, 3rd Edition*. Addison-Wesley-Longman, 2000.

[16] S. Jiang, T. Jiang, and L. Wang, “Secure and efficient cloud data deduplication with ownership management,” *IEEE Transactions on Services Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[17] C. Zuo, J. Shao, J. K. Liu, G. Wei, and Y. Ling, “Fine-grained two-factor protection mechanism for data sharing in cloud storage,” *IEEE Trans. Information Forensics and Security*, vol. 13, no. 1, pp. 186–196, 2018.

[18] J. Li, J. Li, D. Xie, and Z. Cai, “Secure auditing and deduplicating data in cloud,” *IEEE Trans. Computers*, vol. 65, no. 8, pp. 2386–2396, 2016.

[19] R. S. Kumar and A. Saxena, “Data integrity proofs in cloud storage,” in *Third International Conference on Communication Systems and Networks, COMSNETS 2011, Bangalore, India, January 4-8, 2011*, 2011, pp. 1–4.

[20] C. Wang, Q. Wang, K. Ren, and W. Lou, “Privacy-preserving public auditing for data storage security in cloud computing,” in *INFOCOM 2010. 29th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 15-19 March 2010, San Diego, CA, USA*, 2010, pp. 525–533.

[21] Z. Yan, W. Ding, X. Yu, H. Zhu, and R. H. Deng, “Deduplication on encrypted big data in cloud,” *IEEE Trans. Big Data*, vol. 2, no. 2, pp. 138–150, 2016.

[22] D. Boneh, X. Boyen, and E. Goh, “Hierarchical identity based encryption with constant size ciphertext,” in *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, 2005, pp. 440–456.

[23] V. Shoup, “Lower bounds for discrete logarithms and related problems,” in *Advances in Cryptology - EUROCRYPT ’97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, 1997, pp. 256–266.

[24] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[26] D. Comer, “The ubiquitous b-tree,” *ACM Comput. Surv.*, vol. 11, no. 2, pp. 121–137, 1979.

[27] D. Boneh, C. Gentry, and B. Waters, “Collusion resistant broadcast encryption with short ciphertexts and private keys,” in *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, 2005, pp. 258–275.

[28] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[29] S. Goldwasser and S. Micali, “Probabilistic encryption and how to play mental poker keeping secret all partial information,” in *Proceedings of*

the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA, 1982, pp. 365–377.

[30] T. Jiang, X. Chen, Q. Wu, J. Ma, W. Susilo, and W. Lou, "Secure and efficient cloud data deduplication with randomized tag," *IEEE Trans. Information Forensics and Security*, vol. 12, no. 3, pp. 532–543, 2017.

[31] B. Lynn, "Pbc library," https://crypto.stanford.edu/pbc/, 2006.

[32] "Openssl," https://www.openssl.org/.

[33] M. W. Storer, K. M. Greenan, D. D. E. Long, and E. L. Miller, "Secure data deduplication," in *Proceedings of the 2008 ACM Workshop On Storage Security And Survivability, StorageSS 2008, Alexandria, VA, USA, October 31, 2008*, 2008, pp. 1–10.

[34] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," *J. ACM*, vol. 51, no. 2, pp. 231–262, 2004.

[35] D. L. Vo, F. Zhang, and K. Kim, "A new threshold blind signature scheme from pairings," in *Proceedings of the Symposium on Cryptography and Information Security*, 2003, pp. 233–238.

[36] V. Shoup, "Practical threshold signatures," in *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, 2000, pp. 207–220.

[37] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*, 2003, pp. 31–46.

Xue Yang received the B.S. degree in information security from the Southwest Jiaotong University, Chengdu, China, in 2012. She is currently working towards a Ph.D. degree in information and communication engineering, Southwest Jiaotong University. Her research interests include big data security and privacy, applied cryptography and network security.

Rongxing Lu (S'09-M'10-SM'15) received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Waterloo, Canada, in 2012. He was a PostDoctoral Fellow with the University of Waterloo from 2012 to 2013. He was an Assistant Professor with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, from 2013 to 2016. He has been an Assistant Professor with the Faculty of Computer Science, University of New Brunswick, Canada, since 2016. His research interests include applied cryptography, privacy enhancing technologies, and IoT-Big Data security and privacy. He currently serves as the Secretary of the IEEE ComSoc CIS-TC. He is currently a Senior Member of the IEEE Communications Society. He received the most prestigious Governor General's Gold Medal and the 8th IEEE Communications Society (ComSoc) Asia Pacific Outstanding Young Researcher Award, in 2013.

JUN SHAO received the Ph.D. degree from the Department of Computer Science and Engineering at Shanghai Jiao Tong University, Shanghai, China in 2008. He was a postdoc in the School of Information Sciences and Technology at Pennsylvania State University, USA from 2008 to 2010. He is currently a professor of the School of Computer Science and Information Engineering at Zhejiang Gongshang University, Hangzhou, China. His research interests include network security and applied cryptography.

Xiaohu Tang (M'04) received the Ph.D. degree in electronic engineering from the Southwest Jiaotong University, Chengdu, China, in 2001. From 2003 to 2004, he was a research associate in the Department of Electrical and Electronic Engineering, Hong Kong University of Science and Technology. From 2007 to 2008, he was a visiting professor at University of Ulm, Germany. Since 2001, he has been in the School of Information Science and Technology, Southwest Jiaotong University, where he is currently a professor. His research interests include coding theory, network security, distributed storage and information processing for big data.

Dr. Tang was the recipient of the National excellent Doctoral Dissertation award in 2003 (China), the Humboldt Research Fellowship in 2007 (Germany), and the Outstanding Young Scientist Award by NSFC in 2013 (China). He serves as Associate Editors for several journals including IEEE TRANSACTIONS ON INFORMATION THEORY and *IEICE Trans on Fundamentals*, and served on a number of technical program committees of conferences.

ALI A. GHORBANI (SM'–) has held a variety of positions in academia for the past 35 years. He has been the Dean of the Faculty of Computer Science since 2008. He is currently the Canada Research Chair (Tier 1) in Cybersecurity. He is also the Director of the Canadian Institute for Cybersecurity. He has developed a number of technologies that have been adopted by high-tech companies. He co-founded two startups, Sentrant and EyesOver in 2013 and 2015, respectively. He is the Co-Inventor on three awarded patents in the area of network security and web intelligence and has published over 200 peer-reviewed articles during his career. He has supervised over 160 research associates, post-doctoral fellows, and graduate and undergraduate students during his career. His book, *Intrusion Detection and Prevention Systems: Concepts and Techniques*, (Springer, 2010). Since 2010, he has obtained over 10M to fund six large multi-project research initiatives. He was twice one of the three finalists for the Special Recognition Award at the 2013 and 2016 New Brunswick KIRA Award for the knowledge industry. In 2007, he received the University of New Brunswick's Research Scholar Award. He is the Co-Editor-In-Chief of *Computational Intelligence Journal*.