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

Xue Yang (Visiting Student), Rongxing Lu*, Jun Shao, Xiaohu Tang and Ali A. Ghorbani

Contact Email: rlu1@unb.ca

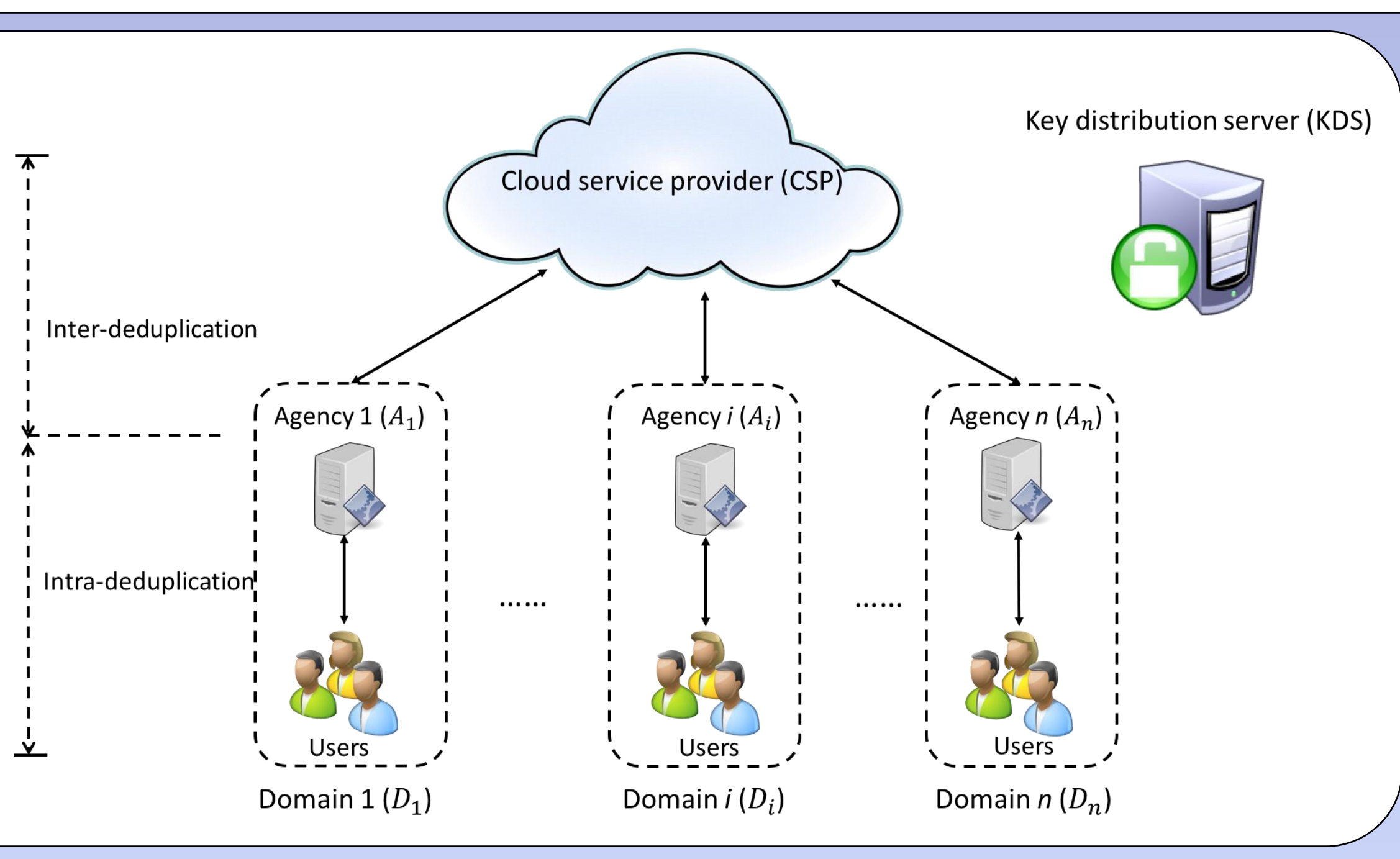
Canadian Institute for Cybersecurity (CIC), University of New Brunswick (UNB)

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 contrary to practical applications. Moreover, few existing schemes consider protecting the linkability information. 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 constant size random ciphertexts for each data, our scheme not only ensures data confidentiality and availability under multi-domain deduplication but also resists brute-force attacks. By allowing only the cloud service provider to perform cross-domain duplication checking over random tags, our scheme can protect the linkability information from disclosure as much as possible. Detailed security analysis shows that our scheme achieves privacy-preservation for both data content and the linkability information, data availability and integrity while resisting brute-force attacks. Furthermore, extensive simulations demonstrate that our scheme significantly outperforms the existing competing scheme, in terms of computational, communication and storage overheads together with the time complexity of the duplicate search.

System Model

- **Key distribution server (KDS):** is responsible for generating private keys for users and a secret for the cloud service provider.
- **Cloud service provider (CSP) :** offers storage services for users and conducts the inter-deduplication over outsourced data.
- **Agency (A_i):** performs intra-deduplication and transforms the intra-tag into a random inter-tag.
- **User:** outsource encrypted data to the CSP for storage.



Design Goals

- The proposed scheme should ensure data confidentiality, availability and integrity while resisting brute-force attacks.
- The proposed scheme should be computation, communication, and storage efficient.

1. System Initialization

The KDS generates the composite bilinear parameters tuple $(N = pq, G, G_T, e)$, and then generates system parameters and private keys as follows.

- Randomly choose two generators $g, x \in G$, and two numbers $\alpha, \gamma \in Z_N$ and then compute $y = x^q$, $v = g^{\gamma}$, $g_i = g^{\alpha^i}$ for $i = 1,2,\ldots,n,n+2,\ldots,2n$.
- For all users in D_i, compute the private key: $d_i = g_i^{\gamma}$.
- Choose three cryptographic hash functions: $h_1: \{0,1\}^* \rightarrow \{0,1\}^{\kappa_1}$, $h_2: \{0,1\}^* \rightarrow \{0,1\}^{\kappa_0 - 1}$ and $h_3: G \rightarrow \{0,1\}^{\kappa_1}$.

Algorithm 1 Search algorithm in the deduplication decision tree

```
Function: Search($L_m$, node)
1: The search function Search($L_m$, node) is started from the root node, i.e., node = root, and it proceeds to a leaf node as follows:
2: if node is a leaf then
3:     return node
4: else
5:     case 1: $L_m < \delta_1$ then
               return Search($L_m, P_0$)
       case 2: $\delta_k \leqslant L_m < \delta_{k+1}$ then
               return Search($L_m, P_k$)
       case 3: $\delta_{n_c-1} \leqslant L_m$ then
               return 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: $(P_0, \delta_1, P_1, \delta_2, P_2, \ldots, \delta_{n_c-1}, P_{n_c-1})$.

2. Data Upload

- **Tag generation:** For uploading the data m, U generates an intra-tag with the private key d_i as $\tau_m = d_i^{h_2(m)}$. Then, U sends a message ``$upload \parallel (L_m, \tau_m)$'' to A_i.
- **Intra-deduplication:** A_i computes $T_m = h_3(\tau_m)$, and checks whether a duplicated exists. If a duplicate exists, U does not need to upload m. Otherwise, A_i computes $\widehat{\tau_m} = \tau_m \cdot y^r$ and sends it to the CSP.
- **Inter-deduplication:** The CSP leverages Algorithm 2 to check whether the duplicate exists or not among different domains.
- **Data encryption / key recovery:** After receiving the feedback, if the message is “data upload”, U performs data encryption operations. If the received message is ``duplication $\parallel B_{m^*}$'', U conducts key recovery operations.

Data encryption: U encrypts the data m before outsourcing as follows.

(1) Randomly choose $\beta_m \in \mathbb{Z}_N$, and set $K_m = e(g_{n+1}, g)^{\beta_m}$, and then generate the convergent key $ck_m = h_1(K_m \parallel m)$.

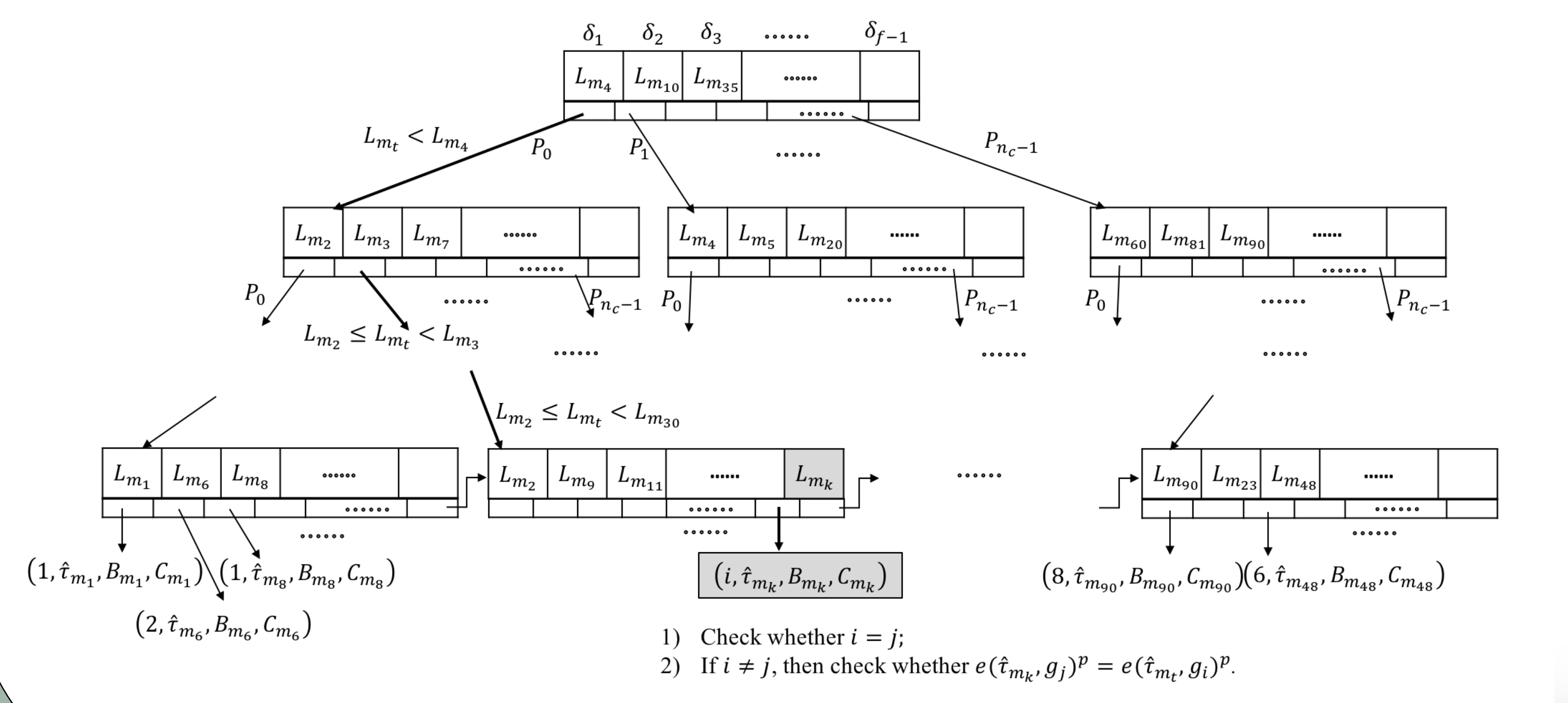
(2) The ciphertext of m can be computed as

$$\begin{cases} C_m = SKE_{ck_m}(m) \\ B_m = \left(g^{\beta_m},\ (v \cdot \prod_{k \in I} g_{n+1-k})^{\beta_m}\right) \end{cases}$$

Key recovery: U recovers the convergent key $ck_{m^*} = h_1(K_{m^*} \parallel m^*)$ with the private key d_i and m as follows.

(1) Let $B_{m^*} = (B_0, B_1)$, compute $e(g_{n+1}, g)^{\beta_{m^*}} = \frac{e(g_i, B_1)}{e\left(d_i \cdot \prod_{k \in I, k \neq i} g_{n+1-k+i},\ B_0\right)}$

(2) Then U recovers ck_{m^*} with m as: $ck_{m^*} = h_1(e(g_{n+1}, g)^{\beta_{m^*}} \parallel m)$.


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}
$L_{m_2} \leq L_{m_t} < L_{m_3}$
$L_{m_2} \leq L_{m_t} < L_{m_{30}}$
$(i, \hat{\tau}_{m_k}, B_{m_k}, C_{m_k})$
1) Check whether $i = j$;
2) If $i \neq j$, then check whether $e(\hat{\tau}_{m_k}, g_j)^p = e(\hat{\tau}_{m_t}, g_i)^p$.

Algorithm 2 Inter-deduplication algorithm

```
1: For the received data $(i, L_m, \hat{\tau}_m)$, the CSP calls the function Search($L_m$, 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 $L_{m^*}$ ($L_{m^*} = L_m$) is found in DDT, e.g., $(j, L_{m^*}, \hat{\tau}_{m^*}, B_{m^*}, C_{m^*})$, 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
               $e(\hat{\tau}_m, g_j)^p = e(\hat{\tau}_{m^*}, g_i)^p$        (1)
10:        if Eq. (1) 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. (1) holds, if and only if $m = m_*$. That is, the duplication can be found by verifying Eq. (1).

3. Data Download

- U recovers m' by decrypting C_m with ck_m.
- Then, U generates the intra-tag $\tau_{m'} = d_i^{h_2(m')}$ and computes its hash value $T_{m'} = h_3(\tau_{m'})$.
- At last, U verifies the integrity by checking whether $T_{m'} = T_m$ holds.