

Optimal Packing for Encrypted and Compressed Key-Value Stores with Pattern-Analysis Security

Chen Zhang, Shujin Ye, Hai Liu, and Tse-Tin Chan, *Member, IEEE*

Abstract—Rising concerns about data privacy and volume have driven the development of encrypted and compressed key-value (KV) storage systems. To defend against pattern-analysis attacks, the length and access frequency distributions of packs should appear uniform to adversaries. The design of the packing algorithm is crucial because it determines both pack length and frequency distributions, thereby impacting the overhead for hiding pack pattern information. Existing algorithms focus on minimizing length differences, leading to large variations in pack frequency and thus causing large bandwidth overhead. In this paper, we study the optimal packing problem for encrypted and compressed KV stores, aiming to minimize the overheads for protecting both pack length and frequency information. We propose DualPacking, a two-dimensional packing algorithm with an approximation ratio that depends on the length and frequency distributions of KV pairs. We further develop an encrypted and compressed KV storage system that adapts well to dynamic updates of outsourced stores. Finally, we formally analyze the security of our design and implement it on Redis and RocksDB. Experimental results indicate that, compared to existing packing algorithms, our design reduces the bandwidth overhead by up to 25% and the storage overhead for pack length protection by 33%, confirming its superior efficiency.

Index Terms—Encryption, compression, key-value store, packing algorithm, pattern-analysis security

I. INTRODUCTION

Key-Value (KV) [1]–[3] stores are widely recognized as high-performance data storage systems, playing a critical role in numerous big data applications [4], [5]. As enterprises increasingly outsource data applications to the cloud for better availability and cost-effectiveness, safeguarding sensitive data on untrusted platforms has gained significant attention. To mitigate privacy risks, data is typically encrypted before being outsourced to the cloud [6]–[8]. Furthermore, as data volumes expand, compression becomes indispensable. It not only reduces storage requirements but also improves processing efficiency by optimizing memory utilization [9], [10]. Therefore, both encryption and compression are essential for the secure and efficient outsourcing of large-scale data to the cloud.

Some encrypted and compressed KV storage systems have been developed to integrate encryption and compression without compromising performance [11]–[14]. In these designs, KV pairs are organized into packs of a certain size, which

are then encrypted and compressed to ensure data privacy and compression efficiency. However, recent studies on secure cloud storage reveal that encrypted data can still be vulnerable. Due to variations in pack length and frequency, adversaries might exploit pattern-analysis attacks, such as frequency analysis [15] and volume attacks [16], to extract sensitive information. The exposure of access patterns, along with prior knowledge of the store, allows adversaries to infer sensitive content within the encrypted and compressed key-value store.

To resist pattern-analysis attacks, it is essential to ensure a uniform distribution of pack lengths and access frequencies [11], [17]. This is typically achieved by padding all packs to a consistent length [11] and injecting fake queries to obscure access frequencies [18], which incurs additional storage and bandwidth overhead for data outsourcing. Minimizing these overheads requires a well-designed packing algorithm, as it directly influences the distribution of pack lengths and frequencies. However, no existing packing algorithms effectively minimize the cost for smoothing both lengths and frequencies simultaneously. Recently, a length-first packing algorithm [19] was proposed to minimize padding overhead by ensuring that pack lengths are closely aligned. Despite its efficiency, this approach neglects frequency during packing, leading to large variations in pack frequency that increase bandwidth overhead and ultimately result in high data outsourcing costs.

In this paper, we focus on the design of encrypted and compressed KV storage systems with pattern-analysis security. We first study the packing problem of KV pairs, aiming to minimize the overall cost of data outsourcing while maintaining pattern-analysis security. To achieve this goal, we analyze the impact of packing algorithms on storage and bandwidth overheads and propose DualPacking, a two-dimensional packing algorithm. DualPacking considers the overhead for smoothing both length and access frequency simultaneously while making packing decisions with global consideration. After that, a comprehensive analysis of DualPacking is provided to evaluate its performance. We detail the construction of our proposed encrypted and compressed KV stores for various operations such as get, put, and delete. The contributions of this paper are summarized as follows:

- We formulate the **Cost-Efficient and Pattern-Protected Packing Problem (CEPPP)** for encrypted and compressed KV stores, aiming to minimize the overhead to achieve pattern-analysis security. To best of our knowledge, we are the first to formally define the problem.
- We analyze the impact of packing algorithm on storage and bandwidth overheads and propose a two-dimensional

C. Zhang and H. Liu are with the Department of Computer Science, The Hong Kong Baptist University of Hong Kong, Hong Kong SAR, China. E-mail: {czhang, hliu}@hkbu.edu.hk.

S. Ye is with the School of Data Science and Engineering, Guangdong Polytechnic Normal University, China. Email: yeshujin@gpnu.edu.cn.

T.-T. Chan is with the Department of Mathematics and Information Technology, The Education University of Hong Kong, Hong Kong SAR, China. E-mail: tsetinchan@eduhk.hk.

packing algorithm, DualPacking, to solve the CEPPP problem. It considers both length and frequency protection during packing.

- We prove that the CEPPP problem is NP-hard and theoretically analyze DualPacking, providing both its upper bound and approximation ratio.
- We develop an encrypted and compressed KV storage system with pattern-analysis security and analyze its security. The proposed system is implemented on Redis and RocksDB. Extensive experimental results demonstrate the high efficiency of our design.

The rest of the paper is organized as follows. Section II presents the system model, threat model, and target problem. Section III formulates the packing problem of KV pairs. In Section IV, we detail the proposed DualPacking algorithm and provide a comprehensive performance analysis. Section V discusses detailed construction of the proposed encrypted and compressed KV store. The security analysis is provided in Section VI. Section VII evaluates the performance of our design. Related work is reviewed in Section VIII. Finally, we conclude the paper in Section IX.

II. PROBLEM DEFINITION

A. System Model

Fig. 1 depicts the architecture of our encrypted and compressed KV storage system, which includes three entities: users, the proxy, and the cloud service provider (CSP).

- **Users:** Users in an institution outsource their data to the CSP via the proxy, allowing them to access it by issuing queries through the proxy.
- **Proxy:** The proxy resides within the same internal network as users, acting as an intermediary between users and the external CSP. Its duties include initializing encrypted and compressed KV stores and handling query execution on behalf of users.
- **Cloud service provider:** The CSP provides cloud storage and data access services to users within the institution.

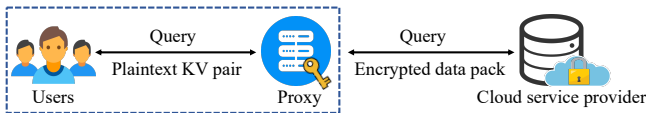


Fig. 1: System architecture

The data uploaded to the proxy is organized into a KV store \mathcal{D} . The proxy initializes \mathcal{D} into an encrypted and compressed KV store $\widehat{\mathcal{D}}$ and outsources $\widehat{\mathcal{D}}$ to the CSP. To ensure efficient compression, the proxy divides KV pairs in \mathcal{D} into packs [11] and then encrypts each pack with its secret key. For ease of data retrieval, each encrypted and compressed pack is assigned a pack identifier (referred to as pid). Users access the data in the CSP via the proxy, utilizing standard KV store operations such as get, put, and delete. The notations used in this paper are summarized in the TABLE I.

As shown in Fig. 1, the data flow between users and the proxy is in plaintext, with data exchanged as KV pairs. The data flow between the proxy and the CSP is in ciphertext, with

TABLE I: Summary of notations.

Notation	Description
$\mathcal{D}, \widehat{\mathcal{D}}$	original KV store, encrypted and compressed KV store
\mathcal{I}	key-pid index maintained by the proxy
\mathcal{P}	set of packs grouped from \mathcal{D}
n	number of KV pairs in \mathcal{D}
m	number of packs in \mathcal{P}
$x_{i,j}$	$x_{i,j} = 1$ if KV pair $i \in \mathcal{D}$ is allocated in pack $j \in \mathcal{P}$; otherwise $x_{i,j} = 0$
l_i, f_i	length, access frequency of KV pair $i \in \mathcal{D}$
L_{\max}	length of the longest pack
F_{\max}	frequency of the pack with the largest access frequency
r	average compression ratio
C_{st}	storage overhead for outsourcing \mathcal{D}
C_{bw}	bandwidth overhead for data access
C	overall cost of data outsourcing
L^*	optimal pack size
β	weight factor of bandwidth overhead
D	user query distribution over keys in \mathcal{D}
D_e	estimated user access distribution maintained by the proxy
L_k	total length of KV pairs assigned to the k -th pack
L_k^1	cumulative length of KV pairs up to the k -th pack
F_k	total frequency of KV pairs assigned to the k -th pack
F_k^1	cumulative frequency of KV pairs up to the k -th pack

data exchanged as encrypted and compressed packs. The proxy maintains a key-pid index \mathcal{I} . When the proxy receives a user query request, it looks up \mathcal{I} and sends the pid corresponding to the queried key to the CSP. The CSP, using the pid, retrieves and sends back the specified data pack to the proxy. The proxy then decrypts and decompresses the pack, extracts the required KV pair, and returns the result to the user.

Suppose user queries on keys in \mathcal{D} follow a distribution D , which can vary over time. Let $D(k)$ represent the probability of querying a specific key k . Consistent with previous designs [17]–[19], we consider the scenario where user queries are drawn independently from D . Since all user queries are routed through the proxy to the CSP, the proxy can maintain an estimate of the user access distribution D_e . With D_e and the key-pid index \mathcal{I} , the access frequency distribution of packs can be calculated by the proxy.

B. Problem Definition

In secure and compressed KV stores, KV pairs are grouped into packs of a certain size, which are subsequently encrypted and compressed. Since KV pairs may vary in length and access frequency, the length and frequency distributions of packs transmitted between the proxy and CSP may be non-uniform, making the store vulnerable to pattern-analysis attacks [19].

In this paper, we focus on designing encrypted and compressed KV stores with pattern-analysis security. The store must ensure that adversaries cannot infer any unauthorized information by observing the traffic between the proxy and the CSP, while minimizing the overall cost of data outsourcing. The overall cost of data outsourcing includes both storage overhead in the cloud and bandwidth overhead for data access. Protecting pack length information incurs additional storage

overhead for data padding, while protecting pack access frequency information results in extra bandwidth overhead for fake query transmission. To minimize these overheads, the design of the packing algorithm is crucial, as it determines the length and frequency distribution of packs. Our goal is to develop a packing algorithm that minimizes the overhead incurred by the protection of pack length and frequency.

C. Threat Model

Consistent with previous designs [12], [18], we assume that the proxy and users are honest and operate within a trusted internal network. The proxy reliably initializes the store outsourced to the CSP, executes user queries, and returns results to users. The CSP is considered to be honest-but-curious, meaning it follows predefined protocols but is curious about the data it hosts. We assume both the CSP and external attackers are passive persistent adversaries with the knowledge of the distribution of key lengths and access frequencies in the outsourced store. They can only eavesdrop on the traffic between the proxy and the CSP, and have no ability to decrypt or modify the transmitted data. Additionally, adversaries cannot query any data in the CSP by controlling the users.

III. PROBLEM FORMULATION

A. Overall Cost of Data Outsourcing

To protect the encrypted and compressed KV stores against pattern-analysis attacks, we need to hide the length and access frequency information of KV pairs, which incurs additional overhead. In this work, we consider two types of overheads associated with designing the encrypted and compressed KV store with pattern-analysis security: 1) storage overhead for storing data in the cloud; and 2) bandwidth overhead for transmitting data between the proxy and the CSP.

Storage overhead: The storage overhead is defined as the total size of the encrypted and compressed KV store outsourced to the cloud. Given a KV store \mathcal{D} with n KV pairs, the proxy groups KV pairs into packs and then compresses each pack. To protect pack length information, all compressed packs are padded to the longest size before encryption. We assume that the encryption operation does not change the data size. Let \mathcal{P} denote the set of packs grouped from \mathcal{D} , which includes m packs. Let $x_{i,j} = \{0, 1\}$ denote the allocation of KV pair $i \in \mathcal{D}$ in the pack $j \in \mathcal{P}$. If KV pair i is assigned to the pack j , then $x_{i,j} = 1$, otherwise $x_{i,j} = 0$. Let l_i denote the length of KV pair i . $L_{max} = \max_{j \in \mathcal{P}} \left(\sum_{i \in \mathcal{D}} x_{i,j} l_i \right)$ represents the length of the longest pack. r denotes the average compression ratio of \mathcal{D} using the compression algorithm c under pack size L_{max} . We assume that the largest pack before compression remains the largest after compression. With r , L_{max} , and m , the cost for storing \mathcal{D} with length protection is defined as

$$C_{st} = \frac{mL_{max}}{r}. \quad (1)$$

Bandwidth overhead: The bandwidth overhead is measured by the amount of data transmitted per user query. Since

KV pairs are stored in encrypted and compressed packs and the CSP has no ability to decrypt the data, the CSP needs to return the entire pack to the proxy for each user query on the key. Meanwhile, because KV pairs in \mathcal{D} may differ in access frequency, the pack access frequency distribution can be non-uniform, potentially leaking the frequency information of these KV pairs. To hide the frequency of packs, the typical method is to include fake queries to disturb the pack frequency distribution and make it uniform. Let f_i denote the expected access frequency of KV pair i . The frequency of the pack with the largest access frequency can be calculated as $F_{max} = \max_{j \in \mathcal{P}} \left(\sum_{i \in \mathcal{D}} x_{i,j} f_i \right)$. Given F_{max} for the dataset \mathcal{D} , considering the protection of pack frequency information, the bandwidth overhead between the proxy and the CSP for data access can be defined as

$$C_{bw} = \frac{mF_{max}L_{max}}{nr}. \quad (2)$$

With C_{st} and C_{bw} , the overall cost of outsourcing the store \mathcal{D} compressed using algorithm c is

$$C = C_{st} + \beta C_{bw} = \frac{nmL_{max} + \beta mF_{max}L_{max}}{nr}, \quad (3)$$

where the weight parameter $\beta > 0$. It is used to adjust the impacts of bandwidth and storage overhead on the overall cost.

B. Problem Formulation

Given a KV store \mathcal{D} outsourced by users, where the length and access frequency information of KV pairs are known, the proxy groups the store into a set of packs \mathcal{P} and compresses each pack using the algorithm c . Additional overhead is incurred to protect the length and frequency information of KV pairs. Our goal is to design a packing algorithm that minimizes the overall cost of outsourcing \mathcal{D} to the cloud while protecting the access patterns of \mathcal{D} . The **Cost-Efficient and Pattern-Protected Packing Problem (CEPPP)** can be defined as below.

Definition 1. *The Cost-Efficient and Pattern-Protected Packing Problem (CEPPP) for outsourcing an encrypted and compressed KV store can be formulated as the following optimization problem:*

$$\min_{x_{i,j}, i \in \mathcal{D}, j \in \mathcal{P}} C \quad (4)$$

$$s.t. \quad L_{max} \leq L^*, \forall j \in \mathcal{P}, \quad (5)$$

$$x_{i,j} \in \{0, 1\}, \forall i \in \mathcal{D}, \forall j \in \mathcal{P}, \quad (6)$$

$$\sum_{j \in \mathcal{P}} x_{i,j} = 1, \forall i \in \mathcal{D}. \quad (7)$$

The constraint corresponding to Eq. 5 specifies that the size of each pack must be smaller than or equal to the optimal pack size, denoted as L^* . Previous studies have discussed the calculation of the optimal pack size [14]. In this paper, we assume that L^* is known for given system parameters. The constraints corresponding to Eq. 6 and Eq. 7 ensure that each KV pair is uniquely assigned to a pack.

Theorem 1. *The optimization problem CEPPP is NP-hard.*

Proof. We prove the NP-hardness of CEPPP by a polynomial-time reduction from the classical 0-1 Knapsack problem, which is known to be NP-complete. The 0-1 Knapsack problem is defined as selecting a subset of items, each with a given positive weight, to fit into a knapsack with limited capacity, such that the total weight of the selected items is maximized without exceeding the capacity. We construct a reduction from an instance of the 0-1 Knapsack problem to a special case of CEPPP under the following constraint assumptions: (1) The weight parameter $\beta = 0$, i.e., restrict the objective to minimizing the maximum pack length without considering the bandwidth overhead; (2) All KV pairs must be allocated to exactly two packs; (3) The total length of all KV pairs is denoted by $L = \sum_{i=1}^n l_i$, where l_i represents the length of KV pair i ; and (4) No single pack has sufficient capacity to accommodate all KV pairs.

Under the formulation, the objective of CEPPP is to partition a set of KV pairs into two packs to minimize the maximum pack length, L_{\max} . It can be observed that minimizing L_{\max} is equivalent to balancing the pack lengths as closely as possible to $L/2$. Given the constraint that all KV pairs must be allocated, minimizing L_{\max} corresponds to identifying a subset of KV pairs whose cumulative length approaches but does not exceed $L/2$. This formulation can be transformed into the 0-1 Knapsack problem: given a collection of items (KV pairs) with positive lengths (weights), determine a subset with maximum total length not exceeding $L/2$. Thus, resolving this restricted instance of CEPPP is at least as computationally complex as solving the 0-1 Knapsack problem, which is NP-complete. Since this reduction is computable in polynomial time, and CEPPP encompasses this special case, we conclude that CEPPP is NP-hard. \square

IV. OPTIMAL PACKING OF KV PAIRS FOR ENCRYPTED AND COMPRESSED KV STORES

A. Design Overview

The key to addressing the CEPPP optimization problem is to design an efficient packing algorithm. The design of packing algorithm affects the distribution of pack lengths and frequencies, thereby affecting the cost incurred for hiding pack lengths and frequencies. The length-first packing algorithm [19] aims to make the lengths of all packs as close as possible, thereby minimizing the overhead of padding all packs to the same length. However, this approach incurs high bandwidth overhead due to the need to inject more fake queries to smooth the frequencies. Similarly, the frequency-first packing can minimize the maximum difference of pack frequencies, which reduces the cost of frequency smoothing but increases the storage cost of padding all packs to the same length. Thus, in the design of packing algorithm, it is critical to consider pack length and frequency simultaneously to achieve an optimal balance between storage and bandwidth overhead.

To minimize the overall cost of data outsourcing, all packs should have small variances over both lengths and frequencies. Given a KV store \mathcal{D} , suppose KV pairs are divided into m packs in total. If we normalize the length and frequency of KV

pairs, the most ideal packing solution is to make the length and frequency of each pack equal to $\frac{1}{m}$. To achieve this goal, a straightforward idea (referred to as a greedy design) is to select, each time, the KV pair that best moves the length and frequency of the current processing pack towards $\frac{1}{m}$. However, such a design does not consider the impact of unassigned KV pairs, making the solution prone to getting stuck in a local optimum and potentially causing KV pairs to be packed into more packs than necessary.

In real-world applications, the frequency of packs usually follows the long-tail distribution [17], i.e., the access frequency of most packs is similar and low. Focusing solely on selecting unassigned KV pairs that move the length and frequency towards $\frac{1}{m}$ most effectively may result in many low-frequency KV pairs being left over, since each pack has a maximum length restriction. This can lead to a significant number of KV pairs with low access frequency remaining unassigned. Consequently, the frequency of several of the last packs may be low, which greatly increases the overhead associated with protecting pack access frequency.

In this work, we propose DualPacking, a two-dimensional packing algorithm designed to minimize the overhead for smoothing pack length and frequency. In DualPacking, we consider the overhead of smoothing both length and frequency simultaneously while making packing decisions with a global consideration. Specifically, we take into account not only the length and frequency of each pack being processed but also the cumulative length and frequency of already packed packs. This approach helps avoid large variances in length and frequency between the later packed packs and the earlier ones. The toy example shown in Fig. 2 demonstrates that, compared to the greedy design, DualPacking can significantly reduce variances in lengths and frequencies among packs.

In the following subsections, we first present the detailed design of DualPacking, followed by a description of the protection methods for pack length and frequency. Finally, we provide an analysis of DualPacking to evaluate its efficiency.

B. DualPacking: Two-Dimensional Packing Algorithm

The goal of DualPacking is to efficiently allocate KV pairs into packs to approximate a target pack length L^* while keeping both the lengths and frequencies of all packs as uniform as possible. To enable better comparison between pack length and frequency, the length and frequency of all KV pairs are first normalized. In DualPacking, KV pairs are assigned one by one to packs, and a new pack is initiated once the cumulative length of the current pack exceeds L^* .

Suppose the current pack being processed is p_k . The core of DualPacking's decision logic lies in the selection of which unassigned KV pair to add to p_k . Two factors are considered during packing: 1) the total lengths and frequencies of KV pairs assigned to p_k , denoted by L_k and F_k , and 2) the cumulative lengths and frequencies of KV pairs already allocated into packs, denoted by L_k^1 and F_k^1 . The consideration of L_k^1 and F_k^1 enables global optimization rather than focusing on a single pack. DualPacking is guided by evaluating two binary conditions: whether $L_k^1 \geq F_k^1$ and whether $L_k \geq F_k$. The

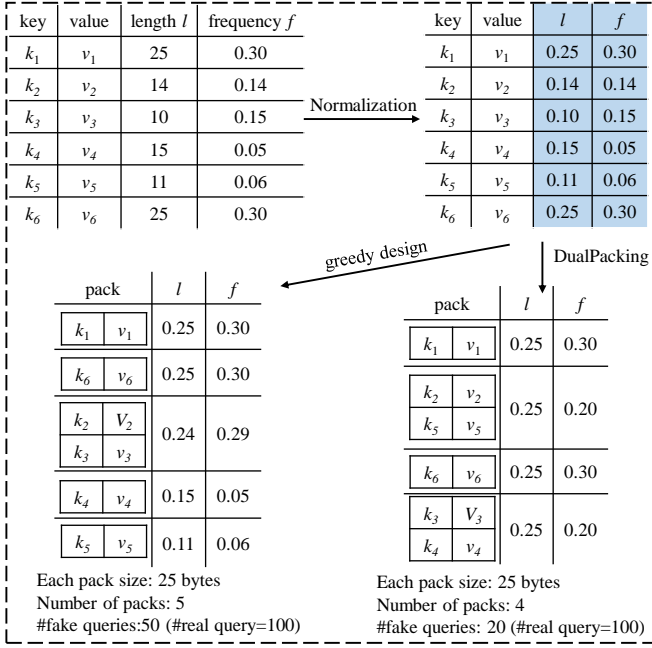


Fig. 2: Toy example of DualPacking.

interplay of these two conditions yields four distinct cases that guide the selection strategy. By evaluating the current case, DualPacking strategically chooses a KV pair that not only progresses p_k towards the target length L^* but also counteracts any existing imbalance between cumulative length and frequency. This ensures that a global balance between total length and frequency is maintained throughout the entire packing process.

Algorithm Description: The details of DualPacking is shown in Alg. 1. Given a KV store \mathcal{D} , we first normalize the length and frequency of all KV pairs separately, as shown in Lines 2-4. DualPacking is structured into two main parts, addressing cases where $L_k^1 \geq F_k^1$ and $L_k^1 < F_k^1$, respectively. Within each of these main parts, we further divide the problem into two subcases based on the values of F_k and L_k . It is important to note that, for each KV pair, two requirements must be met before assigning it to a pack: 1) the KV pair has not been assigned to any pack yet; and 2) the size of the currently processing pack will not exceed L^* after including the KV pair. We now describe DualPacking based on the aforementioned case division.

Case 1: $L_k^1 \geq F_k^1$

- Subcase 1.1: $L_k < F_k$. Let \mathcal{C} denote the set of candidate KV pairs that can be assigned to the currently processing pack p_k . Under this subcase, the KV pairs added to \mathcal{C} must meet two conditions. Firstly, their length values exceed their frequency values. This condition helps minimize the difference between the total length and frequency allocated to the pack. Secondly, after allocation, the relationship between L_k and F_k should remain unchanged, thereby decreasing the difference between L_k^1 and F_k^1 . After determining \mathcal{C} , DualPacking selects the KV pair with the highest length value in \mathcal{C} for allocation.

Algorithm 1: DualPacking

Input: KV store \mathcal{D} , optimal pack size L^* .

Output: The set of packs \mathcal{P} .

```

1  $k \leftarrow 1$ ;  $L_k^1 \leftarrow 0$ ;  $F_k^1 \leftarrow 0$ ;  $\mathcal{R} \leftarrow \mathcal{D}$ ;  $L^* \leftarrow L^*/\sum_{i \in \mathcal{D}} l_i$ ;
2 for each  $i \in \mathcal{D}$  do
   // Normalization
3    $l_i \leftarrow l_i/\sum_{i \in \mathcal{D}} l_i$ ;
4    $f_i \leftarrow f_i/\sum_{i \in \mathcal{D}} f_i$ ;
5 while  $\mathcal{R} \neq \emptyset$  do
6    $Flag \leftarrow True$ ;
7    $L_k \leftarrow 0$ ;  $F_k \leftarrow 0$ ;
8   while  $flag$  do
9      $\mathcal{C} \leftarrow \{i \mid i \in \mathcal{R} \wedge L_k + l_i \leq L^*\}$ ;
10    if  $L_k^1 \geq F_k^1$  then
11      // Case 1
12      if  $L_k < F_k$  then
13        // Subcase 1.1
14         $\mathcal{C} \leftarrow \mathcal{C} \cap \{i \mid i \in \mathcal{R} \wedge (l_i > f_i \vee f_i \leq L^*) \wedge L_k + l_i < F_k + f_i\}$ ;
15        // KV pair candidate set
16      else
17        // Subcase 1.2
18         $\mathcal{C} \leftarrow \mathcal{C} \cap \{i \mid i \in \mathcal{R} \wedge l_i \leq f_i\}$ ;
19      else
20        // Case 2
21        if  $L_k \leq F_k$  then
22          // Subcase 2.1
23           $\mathcal{C} \leftarrow \mathcal{C} \cap \{i \mid i \in \mathcal{R} \wedge l_i > f_i\}$ ;
24        else
25          // Subcase 2.2
26           $\mathcal{C} \leftarrow \mathcal{C} \cap \{i \mid i \in \mathcal{R} \wedge L_k^1 + l_i \geq F_k^1 + f_i\}$ ;
27        if  $\mathcal{C} \neq \emptyset$  then
28          if  $L_k^1 \geq F_k^1$  and  $L_k \geq F_k$  then
29             $i^* \leftarrow \arg \max_{i \in \mathcal{C}} f_i$ ;
30          else
31             $i^* \leftarrow \arg \max_{i \in \mathcal{C}} l_i$ ;
32          Add KV pair  $i^*$  in the  $k$ -th pack of  $\mathcal{P}$ ;
33           $L_k \leftarrow L_k + l_{i^*}$ ;  $F_k \leftarrow F_k + f_{i^*}$ ;
34           $L_k^1 \leftarrow L_k^1 + l_{i^*}$ ;  $F_k^1 \leftarrow F_k^1 + f_{i^*}$ ;
35           $\mathcal{R} \leftarrow \mathcal{R} - \{i^*\}$ ;
36        else
37           $flag \leftarrow false$ ;  $k \leftarrow k + 1$ ;

```

- Subcase 1.2: $L_k \geq F_k$. DualPacking prioritizes selecting the KV pair with the highest frequency value among those where their frequencies exceed their corresponding length values for assignment to a pack.

Case 2: $L_k^1 < F_k^1$

- Subcase 2.1: $L_k \geq F_k$. DualPacking selects the KV pair with the highest length value among those where the length values exceed the frequency values for assignment.
- Subcase 2.2: $L_k < F_k$. Under this subcase, DualPacking selects the KV pair with the highest length value among

those where the sum of their frequency values and F_k is less than the sum of their length values and L_k .

Toy example: Fig. 2 illustrates the process of packing a KV store with 6 KV pairs using DualPacking. The optimal pack size L^* is set to 25. The first step of packing is normalization and L^* is normalized to 0.25. Initially, $L_1^1 = 0$; $F_1^1 = 0$; $\mathcal{R} = \{k_1v_1, k_2v_2, k_3v_3, k_4v_4, k_5v_5, k_6v_6\}$. We start by the construction of pack 1. In the first iteration, since $(L_1^1(0) \geq F_1^1(0)) \& (L_1(0) < F_1(0))$, subcase 1.1 is fulfilled. Thus, the candidate KV pair set $\mathcal{C} = \{k_1v_1, k_6v_6\}$, and the KV pair $\arg \max l_i = \arg \max(l_1, l_6) = k_1v_1$ is assigned to pack 1. Parameters could be updated as $L_1 = 0.25, F_1 = 0.30, L_1^1 = 0.25, F_1^1 = 0.30, \mathcal{R} = \{k_2v_2, k_3v_3, k_4v_4, k_5v_5, k_6v_6\}$.

In the second iteration, the candidate set $\mathcal{C} = \emptyset$, it means pack 1 already capacity full, the packing of pack 1 ends. For the construction of pack 2, initially $L_2 = 0$ and $F_2 = 0$. In iteration 1, $L_2^1(0.25) < F_2^1(0.3) \& L_2(0) \leq F_2(0)$, subcase 2.1 is fulfilled. Then $\mathcal{C} = \{k_4v_4, k_5v_5\}$, $\arg \max l_i = \arg \max(l_4, l_5) = k_4v_4$ is assigned to pack 2. Then $L_2 = 0.15, F_2 = 0.05, L_2^1 = 0.40, F_2^1 = 0.35, \mathcal{R} = \{k_2v_2, k_3v_3, k_5v_5, k_6v_6\}$. In iteration 2, $L_2^1 \geq F_2^1 \& L_2 \geq F_2$, subcase 1.2 is fulfilled. $\mathcal{C} = \{k_3v_3\}$ and then k_3v_3 is assigned to pack 2. $L_2 = 0.25, F_2 = 0.20, L_2^1 = 0.5, F_2^1 = 0.5$. For the next iteration, $\mathcal{C} = \emptyset$, which means the end of the packing of pack 2. Following this way, k_6v_6 is packed in pack 3 and $\{k_5v_5, k_2v_2\}$ are assigned in pack 4. The packing process ends when all KV pairs assigned into packs.

Time complexity We now analyze the worst-case time complexity of Alg. 1. The normalization step requires $O(n)$ time. During each iteration of the outer `while` loop, the algorithm constructs a new pack by iteratively selecting eligible KV pairs from the remaining set. In the worst case, constructing each pack involves scanning all remaining KV pairs to build the candidate set \mathcal{C} and selecting the best item via an `arg max` operation, each requiring $O(n)$ time. Since there are at most n such iterations (each KV pair is packed once), the overall time complexity of the algorithm is $O(n^2)$.

C. Length and Frequency Protection for Packs

With DualPacking, KV pairs are grouped into a set of packs \mathcal{P} . Each pack is compressed and then encrypted. To protect the distribution of pack length, all encrypted and compressed packs are padded to the size of the longest pack among all packs in \mathcal{P} . For the protection of pack frequency distribution, we adopt a randomized process to smooth the distribution to be uniform.

Real queries refer to the user queries on packs, while fake queries are those injected to hide the real query distribution. Let D_e^p and D_f^p denote the real and fake query distribution. Let D_t^p be the target uniformed pack access distribution, where $D_t^p(\text{ID}_P) = \frac{1}{m}$ for all packs $P \in \mathcal{P}$. ID_P is the pid of P . According to [17], real queries and fake queries are mixed in a proportion α ($0 < \alpha \leq 1$) to ensure the following convex combination always holds.

$$\alpha D_e^p + (1 - \alpha) D_f^p = D_t^p. \quad (8)$$

For each user query on a key, the queried key is mapped to a pid according to the key-pid index \mathcal{I} maintained by the

proxy. With the pid, an α -based coin is flipped, which returns heads with probability α . If the coin returns heads, the query is sent to the CSP; otherwise, a query generated from D_f^p is used, and the coin is repeatedly flipped until it returns heads. The value of α affects the bandwidth overhead incurred for protecting pack frequency information. The larger α , the fewer fake queries are injected, resulting in less bandwidth overhead. In our design, we set

$$\alpha = \frac{1}{F_{max} m}, \quad (9)$$

which is the largest value that can ensure D_f^p is always greater than 0 for all packs. According to Eq. 8, D_f^p is computed as

$$D_f^p(\text{ID}_P) = \frac{F_{max} - D_e^p(\text{ID}_P)}{F_{max} m - 1}, \forall P \in \mathcal{P}. \quad (10)$$

Remarks: The frequency protection method described above ensures all packs have uniform access frequency, providing strong protection for the outsourced KV store. Please note that our design can be easily extended to K -indistinguishable frequency smoothing [19] to further reduce the bandwidth overhead at the cost of reduced system security. Users can select the most appropriate smoothing methods based on their specific data security requirements.

D. Algorithm Analysis

In this subsection, we will deduce the lower bound of the proposed DualPacking algorithm. We first analyze pack frequency and deduce its lower bound (to be discussed in Lemma 1). After that, we analyze the number of packs m and provide its upper-bound (discussed in Lemma 2). Based on the analysis results, the approximation ratio of DualPacking is given.

Lemma 1. *For DualPacking, the maximum sum of the frequencies of all KV pairs in a pack F_{max} does not exceed $f_{max} + \frac{\lambda}{\tau} L^*$, where $\lambda = \frac{1}{\sum_{i \in \mathcal{D}} l_i}$, and $\tau = \frac{1}{\sum_{i=1}^n f_i}$, $f_{max} = \max_{i \in \mathcal{D}} \{f_i\}$.*

Proof. We now analyze the processing process of DualPacking. Before packing, we first normalize the lengths and frequencies of KV pairs as numerical values without units, setting $f'_i = \lambda f_i$ and $l'_i = \tau l_i$, which implies $\sum_{i \in \mathcal{D}} f'_i = \sum_{i \in \mathcal{D}} l'_i = 1$. The length constraint for any pack is then defined as $L^{*'} = \lambda L^*$.

Suppose the current pack being processed is pack P_k . Let \mathcal{P}_k be the set of KV pairs assigned to pack P_k , and \mathcal{R}_k be the set of unassigned KV pairs after the assignment of pack P_k , satisfying $\mathcal{R}_{k-1} = \mathcal{R}_k \cup \mathcal{P}_k$. Given a KV pair q with $f'_q > 2L^{*'}$, q is assigned to pack P_k if it meets three conditions: 1) $L_{k-1}^1 \geq F_{k-1}^1$; 2) $f'_q \geq f'_i, \forall i \in \mathcal{R}_{k-1}$; and 3) $l'_i \geq f'_i, \forall i \in \mathcal{P}_k - \{q\}$.

For the first condition, if $L_{k-1}^1 < F_{k-1}^1$, DualPacking ensures $\sum_{i \in \mathcal{P}_k} l'_i \geq \sum_{i \in \mathcal{P}_k} f'_i$ (noting that $L_0^1 = F_0^1 = 0$). Since $\sum_{i \in \mathcal{P}_k} l'_i \leq L^{*'}$, we have $f'_q \leq L^{*'}$, which contradicts $f'_q > 2L^{*'}$. Based on this condition, DualPacking selects the KV pair with the highest frequency from the unassigned KV pairs set \mathcal{R}_{k-1} , making the second condition self-evident. For

the third condition, if $L_{k-1}^1 \geq F_{k-1}^1$, then $L_k \leq F_k$, i.e., $L_{k-1}^1 - F_{k-1}^1 \geq L_k^1 - F_k^1$. Since $\sum_{i \in \mathcal{P}_j} l'_i \leq L^*$, $\forall j \in \mathcal{P}$, we have $L_{k-1}^1 \leq F_{k-1}^1 + L^*$. After assigning KV pair q to pack k , we can obtain:

$$L_{k-1}^1 + \sum_{i \in \mathcal{P}_k} l'_i \leq L_{k-1}^1 + L^* \leq F_{k-1}^1 + L^* + L^* \leq F_{k-1}^1 + f'_q.$$

Thus, any other KV pair assigned to pack k must satisfy $l'_i \geq f'_i$, $\forall i \in \mathcal{P}_k - \{q\}$. Based on the above three conditions, we can deduce

$$f'_q + \sum_{i \in \mathcal{P}_k - \{q\}} f'_i \leq f'_q + L^* \implies \sum_{i \in \mathcal{P}_k} f'_i \leq f'_{\max} + L^*,$$

where $f'_{\max} = \lambda f_{\max}$. Thus, the lower bound of pack frequency using DualPacking is

$$F_{\max} \leq f_{\max} + \frac{\lambda}{\tau} L^*. \quad \square$$

Lemma 2. *The number of packs m in DualPacking does not exceed $\lceil \frac{2 \sum_{i \in \mathcal{D}} l_i}{L^*} - \frac{1}{3}g \rceil$ under the mild condition $l_i \leq \frac{1}{2}L^*$, $\forall i \in \mathcal{D}$, where $g = \sum_{i \in \mathcal{P}'} \lfloor \frac{\tau f_i - \lambda L^*}{\lambda L^*} \rfloor$ and $\mathcal{P}' = \{i \mid f_i \geq 2\frac{\lambda}{\tau}L^*\}$, $\forall i \in \mathcal{D}$.*

Proof. We assume that the KV pairs assigned to each pack $j \in \mathcal{P}$ satisfy $\sum_{i \in \mathcal{P}_j} l'_i \leq \frac{1}{2}L^*$ and $L_j^1 < F_j^1$. Since $\sum_{i \in \mathcal{D}} f'_i = \sum_{i \in \mathcal{D}} l'_i$, there must exist a KV pair q with $l'_q \geq f'_q$ and $q \in \mathcal{R}_j$. Based on the assumption that $l'_q \leq \frac{1}{2}L^*$, KV pair q must be assigned to pack j . It contradicts the fact that $q \in \mathcal{R}_j$. Thus, we conclude that $\sum_{i \in \mathcal{P}_j} l'_i \geq \frac{1}{2}L^*$, $\forall \mathcal{P}_j \subset \mathcal{P} - \mathcal{P}_m$. It can be known that, if change the assumption from $L_j^1 < F_j^1$ to $L_j^1 \geq F_j^1$, the same conclusion still holds.

When assigning a KV pair whose frequency greatly larger than L^* to a pack, subsequent packs must satisfy that the total length of KV pairs assigned to the pack is greater than their total frequency until $L_j^1 \geq F_j^1$, $\forall j \in \mathcal{P}$. Let P_s denote the s -th pack being processed during packing process, where $s \in \{1, 2, \dots, m\}$. Thus, there exists some $s \in \{2, \dots, m-1\}$ such that for each $j \in \{2, \dots, s\}$, the sum of the lengths of the KV pairs in pack \mathcal{P}_j satisfies $\sum_{i \in \mathcal{P}_j} l'_i \geq \sum_{i \in \mathcal{P}_j} f'_i$.

Assuming pack j with $\sum_{i \in \mathcal{P}_j} l'_i < \frac{2}{3}L^*$ and $j \in \{2, \dots, s\}$. According to Alg. 1, there must be a KV pair h with $l'_h \geq f'_h$ and $h \in \mathcal{R}_j$. Because $\sum_{i \in \mathcal{P}_j} l'_i < \frac{2}{3}L^*$, we have $l'_h > \frac{1}{3}L^*$. In the case of $L_{j-1}^1 < F_{j-1}^1$, Alg. 1 assigns KV pairs to the pack in descending order of length. Therefore, pack j contains only one KV pair (otherwise, $\sum_{i \in \mathcal{P}_j} l'_i > \frac{2}{3}L^*$). Since the length of any KV pair is less than $\frac{1}{2}L^*$, we have $\sum_{i \in \mathcal{P}_j} l'_i < \frac{1}{2}L^*$, i.e., KV pair h must be assigned to pack j . This contradicts the conclusion that $h \in \mathcal{R}_j$. Therefore, we have $\sum_{i \in \mathcal{P}_j} l'_i \geq \frac{2}{3}L^*$, $\forall j \in \{2, \dots, s\}$. On the one hand,

$$L_j^1 - F_j^1 = L_{j-1}^1 - F_{j-1}^1 + \sum_{i \in \mathcal{P}_j} l'_i - \sum_{i \in \mathcal{P}_j} f'_i \leq L_{j-1}^1 - F_{j-1}^1 + L^*.$$

On the other hand,

$$F_1 - L_1 = \sum_{i \in \mathcal{P}_1} f'_i - \sum_{i \in \mathcal{P}_1} l'_i \geq f'_{\max} - L^*.$$

We can obtain $s \geq \lfloor \frac{f'_{\max} - L^*}{L^*} \rfloor$. Thus, there are at least g packs, and the lengths of the KV pairs assigned to them

are greater than $\frac{2}{3}L^*$, where $g = \sum_{i \in \mathcal{P}'} \lfloor \frac{f'_i - L^*}{L^*} \rfloor = \sum_{i \in \mathcal{P}'} \lfloor \frac{\tau f_i - \lambda L^*}{\lambda L^*} \rfloor$, and $\mathcal{P}' = \{i \mid f'_i \geq 2L^*\} = \{i \mid f_i \geq 2\frac{\lambda}{\tau}L^*\}$, $\forall i \in \mathcal{D}$.

Based on the above analysis, we have

$$\begin{aligned} & \frac{1}{2}(m-1-g)L^* + \frac{2}{3}gL^* < \sum_{i \in \mathcal{D}} l_i \\ \implies m & < \frac{2 \sum_{i \in \mathcal{D}} l_i}{L^*} - \frac{1}{3}g + 1 \leq \left\lceil \frac{2 \sum_{i \in \mathcal{D}} l_i}{L^*} - \frac{1}{3}g \right\rceil. \quad \square \end{aligned}$$

Theorem 2. *Our algorithm is an approximation algorithm with an approximation ratio of $\mu \left(1 + \frac{1}{\beta f_{\max} + 1}\right)$ under the mild condition $l_i \leq \frac{1}{2}L^*$, for all $i \in \mathcal{D}$, where $\mu = 2 - \frac{gL^*}{3 \sum_{i \in \mathcal{D}} l_i} + \frac{L^*}{\sum_{i \in \mathcal{D}} l_i}$.*

Proof. Let \mathbf{X}^* be the optimal solution, with $z(\mathbf{X}^*)$ representing the function that returns the objective function value of \mathbf{X}^* . Denote the number of packs in the optimal solution by m^* and the maximum sum of the frequencies of KV pairs placed in the same pack in the optimal solution by F^* . Let $m' = \frac{\sum_{i \in \mathcal{D}} l_i}{L^*}$. We have:

$$\begin{aligned} z(\mathbf{X}^*) &= \frac{nm^*L^* + \beta m^*F^*L^*}{nr} \\ &= \frac{m^*L^*(n + \beta F^*)}{nr} \\ &\geq \frac{\sum_{i \in \mathcal{D}} l_i (n + \beta f_{\max})}{nr}. \end{aligned}$$

We also have:

$$\begin{aligned} z(\mathbf{X}) &\leq \frac{mL^*}{r} + \frac{\beta m \bar{F}_{\max} L^*}{nr} \\ &\leq \frac{\mu m' L^* (n + \beta \bar{F}_{\max})}{nr}. \end{aligned}$$

Therefore, we have:

$$\begin{aligned} \frac{z(\mathbf{X})}{z(\mathbf{X}^*)} &\leq \frac{\mu m' L^* (n + \beta \bar{F}_{\max})}{\sum_{i \in \mathcal{D}} l_i (n + \beta f_{\max})} \\ &= \frac{\mu (n + \beta \bar{F}_{\max})}{(n + \beta f_{\max})} \\ &\leq \mu + \frac{\frac{\lambda}{\tau} \mu \beta L^*}{(n + \beta f_{\max})} \end{aligned}$$

In our problem, f_{\max} is much larger than $\frac{\sum_{i \in \mathcal{D}} f_i}{m'}$. Thus,

$$\begin{aligned} \frac{z(\mathbf{X})}{z(\mathbf{X}^*)} &\leq \mu + \frac{\frac{\lambda}{\tau} \mu \beta L^*}{(n + \beta f_{\max})} \\ &= \mu + \frac{\frac{\sum_{i \in \mathcal{D}} f_i}{m'} \mu \beta}{(n + \beta f_{\max})} \\ &\leq \mu \left(1 + \frac{1}{\frac{n}{\beta f_{\max}} + 1} \right). \quad \square \end{aligned}$$

V. IMPLEMENTATION OF ENCRYPTED AND COMPRESSED KV STORES WITH PATTERN-ANALYSIS SECURITY

In this section, we detail the construction of the proposed encrypted and compressed KV storage system. We first introduce the design for handling static stores. Then, we extend the

Algorithm 2: Initialization

Input: KV store \mathcal{D} , estimated user access distribution D_e , secret key of the proxy sk , encryption algorithm Enc, compression algorithm Compress, optimal pack size L^* .

Output: Real query distribution D_e^p over pid, fake query distribution D_f^p over pid, parameter α , key-pid index \mathcal{I} , encrypted and compressed KV store $\widehat{\mathcal{D}}$.

- 1 Call Alg. 1 to group KV pairs into a set of packs \mathcal{P} ;
- 2 **for** $P \in \mathcal{P}$ **do**
- 3 Assign a unique number ID_P as pid;
- 4 $\widehat{C}_P \leftarrow \text{Enc}(sk, \text{Compress}(P))$;
- 5 $D_e^p(ID_P) = 0$;
- 6 **for each key** k **in** P **do**
- 7 Add (k, ID_P) into key-pid index \mathcal{I} ;
- 8 $D_e^p(ID_P) \leftarrow D_e^p(ID_P) + D_e(k)$;
- 9 Find the longest pack length $L' \leftarrow \max_{P \in \mathcal{P}} |\widehat{C}_P|$;
- 10 **for** $P \in \mathcal{P}$ **do**
- 11 Pad \widehat{C}_P to length L' ;
- 12 Add (ID_P, \widehat{C}_P) into index $\widehat{\mathcal{D}}$;
- 13 Outsource $\widehat{\mathcal{D}}$ to the CSP;
- 14 Calculate α according to Eq. 9;
- 15 Calculate D_f^p according to Eq. 10;

design to support dynamic KV stores, incorporating put and delete operations while maintaining pattern-analysis security.

A. Construction of Our Design: Static Case

Initialization: Given a KV store \mathcal{D} with known length and access distribution, packing is the first step to initialize the store outsourced to the CSP. The pack length and frequency are determined based on the KV pairs assigned to each pack. After packing, each pack is compressed and then encrypted. The outsourced store $\widehat{\mathcal{D}}$ is in the form of pid and encrypted and compressed packs. To protect the pack length distribution, all packs are padded to the longest pack size after compression, L' , before outsourcing. The proxy maintains the key-pid index \mathcal{I} and calculates the parameter α and fake query distribution D_f^p to safeguard the pack frequency distribution.

GetKey: As shown in Alg. 3, when the proxy receives a user query request for a key, it first searches the key-pid index \mathcal{I} to find the corresponding pid. The query is then added to the query queue \mathcal{Q} , which holds all real user queries to be processed. Once the queried pack is received from the CSP, the proxy decrypts and decompresses it to obtain the plaintext pack, returning the value associated with the queried key to the user.

The proxy follows a randomized process to mix real query in \mathcal{Q} with fake queries, ensuring the pack frequency distribution observed by adversaries is uniform. As shown in Alg. 4, for each query in \mathcal{Q} , the proxy flips an α -biased coin, denoted by *coin*. If *coin* = 1 (the coin returns heads), the proxy sends the real query to the CSP; otherwise, it sends a

Algorithm 3: GetKey

Input: User query for key k , query queue \mathcal{Q} , decryption algorithm Dec, decompression algorithm Decompress.

Output: Matched value v .

- 1 $ID_P \leftarrow \mathcal{I}[k]$; Add ID_P to \mathcal{Q} ;
- 2 The proxy follows Alg. 4 to send queries to the CSP;
- 3 Wait the CSP returns the query result (ID_P, \widehat{C}_P) ;
- 4 $P \leftarrow \text{Decompress}(\text{Dec}(sk, \widehat{C}_P))$;
- 5 $v \leftarrow P[k]$;

fake query sampled from D_f^p and continues flipping the coin until it returns heads. In the existing design, the last query must be a real query, potentially leaking pattern information. To address this, consistent with [17], for the last query in \mathcal{Q} , the proxy randomly selects a batch size r from the batch set $\mathcal{B} = \{\frac{1}{\alpha}, \frac{1}{\alpha} + 1, \frac{1}{\alpha + 2}\}$ and sends r queries to the CSP, thereby protecting query boundary information.

B. Construction of Our Design: Dynamic Case

DeleteKey: Alg. 5 illustrates the process of delete operations. To protect pack length information, deleted packs are not directly removed from the outsourced store. The proxy maintains an index \mathcal{T} , which maps a pid to a set of deleted keys in the pack. When a KV pair (k_d, v_d) is deleted, the proxy identifies its pid ID_{P_d} and adds the key to $\mathcal{T}[ID_{P_d}]$. Although we do not physically delete (k_d, v_d) from the outsourced store, the key will not be queried after deletion. Therefore, we update the real query distribution of ID_{P_d} by subtracting the access frequency of k_d . When the access distribution varies, the fake query distribution D_f^p and parameter α can be easily adjusted to ensure that the pack access distribution remains uniform. For the low utilization problem caused by repeated delete operations, an efficient solution is to merge non-full packs. By setting a threshold for packs, once the total length of deleted KV pairs exceeds the threshold, non-full packs can be merged to improve storage utilization.

PutKey: Alg. 6 outlines the procedures of the put operation. Upon receiving the KV pair (k', v') , the proxy first verifies its presence in the outsourced store. If the KV pair exists, the proxy proceeds to update the corresponding pack value accordingly. As shown in lines 3-4, if k' has been deleted in prior operations, it is imperative to remove k' from the deletion index \mathcal{T} to maintain data consistency. In cases where (k', v') is absent from the key-pid index \mathcal{I} , (k', v') is added to S , a set of KV pairs to be inserted into the outsourced store. The proxy maintains S . Once the total length of KV pairs in S , denoted by L_{sum} exceeds $2L_{max}$, we utilize DualPacking algorithm to pack KV pairs into packs. This method ensures that the first two packs best match our goal of minimizing the overhead for protecting pack length and frequency information, while also limiting storage overhead in the proxy for storing S . The KV pairs successfully packed in this round are removed from S and the remaining KV pairs in S will wait for the next round of packing.

Algorithm 4: QueryProcessor

Input: Query queue \mathcal{Q} , real query distribution D_e^p , fake query distribution D_f^p , α .

Output: Query requests for ID_{P_q} .

```

1 while  $\mathcal{Q} \neq \emptyset$  do
2   if  $|\mathcal{Q}| > 1$  then
3     while 1 do
4        $coin \xleftarrow{\alpha} \{0, 1\}$ ;
5       if  $coin = 1$  then
6         // Real query
7          $ID_P \leftarrow \text{Dequeue}(\mathcal{Q})$ ;
8         Call  $\text{CSP.GetPK}(ID_P)$ ; break;
9       else
10        // Fake query
11         $ID_P \xleftarrow{\$} D_f^p$ ; Call  $\text{CSP.GetPK}(ID_P)$ ;
12        continue;
13   else
14     // The last query in  $\mathcal{Q}$ 
15      $r \xleftarrow{\$} \mathcal{B}$ ;
16     for  $i = 1, \dots, r$  do
17        $coin \xleftarrow{\beta} \{0, 1\}$ ;
18       if  $coin = 1$  and  $\mathcal{Q} = \emptyset$  then
19          $ID_P \xleftarrow{\$} D_e^p$ ;
20       else if  $coin = 1$  and  $\mathcal{Q} \neq \emptyset$  then
21          $ID_P \leftarrow \text{Dequeue}(\mathcal{Q})$ ;
22       else if  $coin = 0$  then
23          $ID_P \xleftarrow{\$} D_f^p$ ;
24         Call  $\text{CSP.GetPK}(ID_P)$ ;
25    $\widehat{C}_P \leftarrow \widehat{D}[ID_P]$ ;
26   Send  $(ID_P, \widehat{C}_P)$  to the proxy;
```

Remarks: The proxy serves as the portal between users and the CSP. Upon detecting changes in the real query distribution, it can update α and the fake query distribution accordingly. This allows the storage system to efficiently adapt to changes in key access distribution. After an extended period, if the outsourced storage experiences significant changes, the proxy can reinitialize the system to maintain high efficiency.

VI. SECURITY ANALYSIS

Consistent with the formal security model in [17], [19], we consider a persistent passive adversary capable of accessing transcripts of encrypted queries and responses between the proxy and the CSP. This adversary can discern the pack length and access distributions from the intercepted data. Our encrypted and compressed KV store includes protection against pack length and frequency pattern analysis. We now formally define the system security model and provide proof with respect to the two aspects.

Algorithm 5: DeleteKey

Input: Secret key sk , deleted KV pair (k_d, v_d) , index \mathcal{T} .

Output: Index \mathcal{T} .

```

1  $ID_{P_d} \leftarrow \mathcal{I}[k_d]$ ;
2 if  $ID_{P_d}$  not in  $\mathcal{T}$  then
3    $\mathcal{T}(ID_{P_d}) \leftarrow \{k_d\}$ ;
4 else
5   Add  $k_d$  to  $\mathcal{T}[ID_{P_d}]$ ;
6  $D_e^p(ID_{P_d}) \leftarrow D_e^p(ID_{P_d}) - D_e(k_d)$ ;
7 Update  $\alpha$  according to Eq. 9;
8 Update  $D_f^p$  according to Eq. 10;
```

A. Protection Against Pack Length Pattern Analysis

To protect the length patterns of KV pairs from length pattern attacks, all packs are padded to a uniform size. Let $\mathcal{L} = (\mathcal{L}_{init}, \mathcal{L}_{lp}, \mathcal{L}_{up})$ represent the group of leakage functions, where \mathcal{L}_{init} , \mathcal{L}_{lp} , and \mathcal{L}_{up} are the leakage functions for system initialization, length pattern, and update pattern, respectively. \mathcal{L}_{lp} pertains to the leaked length pattern during data accesses. \mathcal{L}_{up} involves the pid and the corresponding pack length leaked during update operations.

Let ECKV = (Initialization, GetKey, QueryProcessor, DeleteKey, PutKey) be the encrypted and compressed KV store scheme. Given a Probabilistic Polynomial Time (PPT) adversary \mathcal{A} , a PPT simulator \mathcal{S} , and the encryption secure parameter λ , we introduce two interactive probabilistic experiments: $\mathbf{Real}_{\mathcal{A}}(1^\lambda)$ and $\mathbf{Ideal}_{\mathcal{A}, \mathcal{S}}(1^\lambda)$, defined as below.

$\mathbf{Real}_{\mathcal{A}}(1^\lambda)$: \mathcal{A} selects a KV store \mathcal{D} and gives it to a challenger. The challenger transforms \mathcal{D} into an encrypted and compressed KV store using the Initialization algorithm. Meanwhile, the challenger can input and delete KV pairs using the PutKey and DeleteKey algorithms. \mathcal{A} makes a polynomial number of queries on keys, and the challenger returns ciphertexts by executing the GetKey and QueryProcessor algorithms. Finally, \mathcal{A} returns a bit, which is the output of the experiment.

$\mathbf{Ideal}_{\mathcal{A}, \mathcal{S}}(1^\lambda)$: \mathcal{A} provides the KV store \mathcal{D} to \mathcal{S} . \mathcal{S} simulates the encrypted and compressed KV store based on \mathcal{L}_{init} . With the update leakage function \mathcal{L}_{up} , \mathcal{S} simulates the insertion or deletion of KV pairs. \mathcal{A} performs a polynomial number of queries on keys. \mathcal{S} responses with ciphertexts, guided by the leakage function \mathcal{L}_{lp} . \mathcal{A} returns a bit as the final output of the experiment.

Definition 2. We consider ECKV is secure against length pattern analysis attacks if for all PPT adversaries \mathcal{A} , there exists a PPT simulator \mathcal{S} such that:

$$|\Pr[\mathbf{Real}_{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Ideal}_{\mathcal{A}, \mathcal{S}}(1^\lambda) = 1]| \leq \text{negl}(1^\lambda),$$

where $\text{negl}(1^\lambda)$ is a negligible function.

Formally, the leakage functions $\mathcal{L}_{init}, \mathcal{L}_{lp}, \mathcal{L}_{up}$ can be cap-

Algorithm 6: PutKey

Input: Secret key sk , newly inserted KV pair (k', v') , KV pair set S , longest pack size L_{max} , longest pack size after compression L' .

Output: KV pair set S .

```

1 if  $k' \in \mathcal{I}$  then
2    $ID_{P_{k'}} \leftarrow \mathcal{I}[k'];$ 
3   if  $k'$  in  $\mathcal{T}[ID_{P_{k'}}]$  then
4     // remove from deletion index  $\mathcal{T}$ 
4     Delete  $k'$  from  $\mathcal{T}[ID_{P_{k'}}]$ ;
5     // Value update
5      $(ID_{P_{k'}}, \widehat{C}_{P_{k'}}) \leftarrow \text{CSP.GetPk}(ID_{P_{k'}})$ ;
6      $P_{k'} \leftarrow \text{Decompress}(\text{Dec}(sk, \widehat{C}_{P_{k'}}))$ ;
7      $P_{k'}[k'] \leftarrow v'$ ;
8      $\widehat{C}'_{P_{k'}} \leftarrow \text{Enc}(sk, \text{Compress}(P_{k'}))$ ;
9     Call  $\text{CSP.UpdatePk}(ID_{P_{k'}}, \widehat{C}'_{P_{k'}})$ ;
10 else
11   Add  $(k', v')$  into  $S$ ;
12  $L_{sum} = \sum_{i \in S} l_i$ ;
13 if  $L_{sum} \geq 2L_{max}$  then
14   Call Alg. 1 to group KV pairs in  $S$  into packs;
15   Add the first two packs into  $\mathcal{P}'$ ;
16   for each pack  $P \in \mathcal{P}'$  do
17     Assign a unique number  $ID_P$  as pid;
18      $\widehat{C}_P \leftarrow \text{Enc}(sk, \text{Compress}(P))$ ;
19      $D_e^p(ID_P) = 0$ ;
20     for each KV pair  $(k, v)$  in  $P$  do
21       Delete  $(k, v)$  in  $S$ ;
22       Add  $(k, ID_P)$  into key-pid index  $\mathcal{I}$ ;
23        $D_e^p(ID_P) \leftarrow D_e^p(ID_P) + D_e(k)$ ;
24     Pad  $\widehat{C}_P$  to length  $L'$ ;
25     Call  $\text{CSP.PutPk}(ID_P, \widehat{C}_P)$ ;
26   Update  $\alpha$  according to Eq. 9;
27   Update  $D_f^p$  according to Eq. 10;
28  $\text{CSP.UpdatePk}(ID_P, \widehat{C}'_P)$ 
29  $\widehat{\mathcal{D}}[ID_P] \leftarrow \widehat{C}'_P$ ;
30  $\text{CSP.PutPk}(ID_P, \widehat{C}_P)$ 
31 Add  $(ID_P, \widehat{C}_P)$  into index  $\widehat{\mathcal{D}}$ ;
```

tured by an external adversary \mathcal{A} are defined as

$$\begin{aligned} \mathcal{L}_{init} &= \{m, (ID_P, L'), ID_P \in \mathcal{P}\}, \\ \mathcal{L}_{lp} &= \{(ID_P, L'), ID_P \in \mathcal{S}_Q\}, \\ \mathcal{L}_{up} &= \{opt, (ID_P, L'), ID_P \in \mathcal{S}_P\}. \end{aligned}$$

m is the number of packs. \mathcal{P} is the set of packs, L' is the longest pack size after compression. \mathcal{S}_Q denotes the set of pack identifies corresponding to keys in query queue Q . \mathcal{S}_P denotes the set of pack identifies affected by update operations. In term of opt in \mathcal{L}^{up} , we only consider the put operation since the delete operation will not cause query to the CSP, as discussed in Section V-B.

Theorem 3. ECKV is \mathcal{L} -secure against length pattern analysis attacks under the random oracle model for the Authenticated Encryption with Associated Data (AEAD) scheme framework.

Proof. An AEAD scheme is secure against Chosen Ciphertext Attack (CCA) when the confidentiality of the proxy's secret key is ensured. We aim to prove the security of the proposed pack length padding scheme under the AEAD scheme framework. To achieve this, we need to demonstrate the existence of a PPT simulator \mathcal{S} such that, for every PPT adversary, the outputs of the experiments $\text{Real}_{\mathcal{A}}(1^\lambda)$ and $\text{Ideal}_{\mathcal{A}, \mathcal{S}}(1^\lambda)$ are computationally indistinguishable. Given the leakage of \mathcal{L}_{init} , the simulator \mathcal{S} simulates all encrypted pack values. These values cannot be distinguished from those of the actual encrypted database. Specifically, \mathcal{S} initializes a store with m entries, where each entry is in the form a pid and a L' -bit random string. Given the leakage of \mathcal{L}_{lp} , \mathcal{S} simulates queries for pid and obtains the corresponding encrypted and compressed pack as real ones. \mathcal{S} follows the random oracle to simulate the data access process over the simulated store. For each query, \mathcal{S} selects targeted pids to match simulated encrypted values, and the length of all returned ciphertexts are the same. For the put operation, if the KV pair is newly added in the store, it will be cached in the proxy. When the total size of cached KV pairs greater than $2L_{max}$, leakages are the same as in \mathcal{L}_{init} . If it is to update the value of a key, leakages are captured in \mathcal{L}_{up} . \mathcal{S} gets and updates the pack as observed. It will also updates the pack with the newly simulated ciphertext. The uniform size of the packs and CCA-security of AEAD ensures that \mathcal{A} is unable to distinguish between real interactions and simulated ones. Then outputs from experiments $\text{Real}_{\mathcal{A}}(1^\lambda)$ and $\text{Ideal}_{\mathcal{A}, \mathcal{S}}(1^\lambda)$ are computationally indistinguishable. \square

B. Protection Against Pack Frequency Pattern Analysis

We now define the security model for the proposed frequency protection scheme. The persistent passive adversary \mathcal{A} is assumed to observe the queries and responses of the packs and to track the distribution of pack access. Using the information, \mathcal{A} attempts to infer sensitive details about the outsourced store, such as key access patterns and key identities. To provide a formal security analysis about our pack frequency protection scheme, we introduce the security definition called Pack-Level Real-Or-Random Indistinguishability under Chosen Distribution Attack (P-ROR-CDA) through the real and ideal games $\text{Game}_{Q, D^p, D_e^p}^{\mathcal{A}}$ and $\text{Game}_Q^{\mathcal{A}}$, defined in Fig. 3. The procedures Initialization and QueryProcessor are defined in Alg. 2 and Alg. 4, respectively. τ is the transcript of a series of queries. \mathcal{A}_1 and \mathcal{A}_2 are challenger and adversary in each game, respectively.

We evaluate the success of an adversary \mathcal{A} in attacking the proposed design by assessing its ability to distinguish between the two games: $\text{Game}_{Q, D^p, D_e^p}^{\mathcal{A}}$ and $\text{Game}_Q^{\mathcal{A}}$. The real game $\text{Game}_{Q, D^p, D_e^p}^{\mathcal{A}}$ is parameterized by the number of queries Q , real pack access distribution D^p , and estimated pack access distribution D_e^p . D^p, D_e^p can be calculated based on D, D_e . In game $\text{Game}_{Q, D^p, D_e^p}^{\mathcal{A}}$, \mathcal{A} has access to the encrypted and

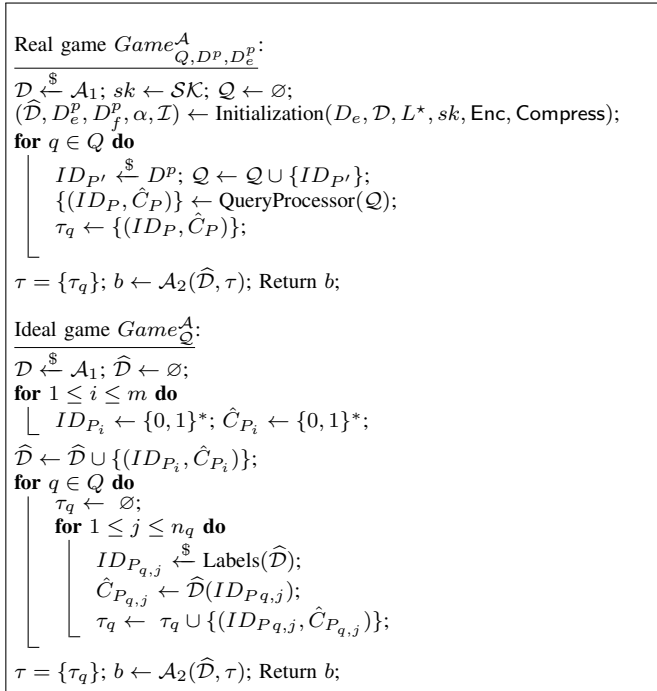


Fig. 3: Security game for pack frequency protection scheme.

compressed KV store along with a transcript of accesses. In $Game_{\mathcal{Q}}^A$, \mathcal{A} has access to a randomly generated encrypted and compressed store and a transcript of randomly chosen queries.

Definition 3. Given the number of queries Q , distributions D^p , D_e^p , our scheme achieves P-ROR-CDA security if the advantage of the adversary \mathcal{A} in breaking games, $Game_{Q, D^p, D_e^p}^A$ and $Game_{\mathcal{Q}}^A$, is negligible, where the advantage $\text{Adv}^{p\text{-ror-cda}}(\mathcal{A})$ is defined as

$$\text{Adv}^{p\text{-ror-cda}}(\mathcal{A}) = |\Pr[Game_{Q, D^p, D_e^p}^A \Rightarrow 1] - \Pr[Game_{\mathcal{Q}}^A \Rightarrow 1]|.$$

We now use Theorem 4 to prove that our design is P-ROR-CDA secure.

Theorem 4. Given the query queue Q and distributions D^p , D_e^p , our design can achieve P-ROR-CDA security for any Q -query if \mathcal{A} 's advantage $\text{Adv}^{p\text{-ror-cda}}(\mathcal{A})$ in breaking $Game_{Q, D^p, D_e^p}^A$ and $Game_{\mathcal{Q}}^A$ is negligible, namely

$$\text{Adv}^{p\text{-ror-cda}}(\mathcal{A}) \leq \text{Adv}_{\text{Enc}}^{\text{ror}}(\mathcal{B}) + \text{Adv}_{Q, D^p, D_e^p}(\mathcal{C}),$$

where Enc is the authenticated encryption scheme adopted by our design. \mathcal{B} is an adversary in the security model of Enc, and \mathcal{C} is a distribution distinguisher.

Proof. The proof is reduced to the securities of real-versus-random indistinguishability of Enc and the indistinguishability of D^p and D_e^p . We define two games G_1 and G_2 . Let G_1 be $\text{Adv}^{p\text{-ror-cda}}(\mathcal{A})$ except that the encryption algorithm Enc is replaced with a random function outputting encrypted strings. The advantage of \mathcal{A} in $\text{Adv}^{p\text{-ror-cda}}(\mathcal{A})$ and G_1 can be upper bounded by the advantage of \mathcal{B} against Enc:

$$|\Pr[Game_{Q, D^p, D_e^p}^A \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| \leq \text{Adv}_{\text{Enc}}^{\text{ror}}(\mathcal{B}).$$

The game G_2 is the same as G_1 except that the distribution D_e^p is replaced by D^p . A reduction gives that

$$|\Pr[G_1 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| \leq \text{Adv}_{Q, D^p, D_e^p}(\mathcal{C}).$$

Based on above discussions, we have

$$|\Pr[Game_{Q, D^p, D_e^p}^A \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| \leq \text{Adv}_{\text{Enc}}^{\text{ror}}(\mathcal{B}) + \text{Adv}_{Q, D^p, D_e^p}(\mathcal{C}).$$

Then the difference between \mathcal{A} 's advantages in G_2 and $Game_{\mathcal{Q}}^A$ is

$$|\Pr[Game_{\mathcal{Q}}^A \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| \leq \text{negl}(1^\lambda).$$

According to above discussions, we can conclude that $\text{Adv}^{p\text{-ror-cda}}(\mathcal{A}) \leq \text{Adv}_{\text{Enc}}^{\text{ror}}(\mathcal{B}) + \text{Adv}_{Q, D^p, D_e^p}(\mathcal{C})$. It can be known that G_2 is distributed identically to $Game_{\mathcal{Q}}^A$, where all encrypted packs are random strings. We now prove that all packs have the same access frequency.

Our design achieves P-ROR-CDA security if the pack access frequency captured by \mathcal{A} is independent to its pid. In our design, each access is independent and sampled from D^p with probability α and or D_f^p with $1 - \alpha$. By constructing the scheme as outlined in Eq. 8, the probability of accessing any pack remains uniform. Let τ' be a random variable denoting the output of the QueryProcessor on an input sampled from D^p . τ'_i is the i -th access in the output. Let (ID_P, \hat{C}_P) is any random encrypted pair. For all i and any (ID_P, \hat{C}_P) , we have

$$\begin{aligned} \Pr[\tau'_i = ((ID_P, \hat{C}_P))] &= \Pr[\tau'_i | \text{coin} = 1] \cdot \alpha + \Pr[\tau'_i | \text{coin} = 0] \cdot (1 - \alpha) \\ &= D^p(ID_P) \cdot \frac{1}{F_{\max} m} + \frac{F_{\max} - D^p(ID_P)}{F_{\max} m - 1} \cdot (1 - \frac{1}{F_{\max} m}) \\ &= \frac{1}{m}. \end{aligned}$$

This proves the independence of the packs. The proof is completed. \square

VII. EXPERIMENTAL EVALUATION

A. Prototype Implementation

The experiment is simulated on a personal computer equipped with 24GB of memory and a 16-core i7 CPU. Both the proxy and CSP are deployed on this computer. The link between the proxy and CSP is set to 1Gbps. The encryption and compression algorithms adopted are Sodium¹ and Zstandard², respectively.

We compare our design with two state-of-the-art encrypted and compressed KV stores: LFP [19] and MiniCrypt [11]. LFP incorporates pack length and frequency protection, prioritizing the smoothing of pack length distribution. In contrast, MiniCrypt organizes KV pairs into packs based on key order, focusing solely on pack length protection without addressing access frequency. We enhance MiniCrypt by adjusting the pack access distribution to be uniform. Our performance comparison spans two representative storage backends: Redis,

¹<https://github.com/winlibs/libsodium>

²<https://github.com/facebook/zstd>

TABLE II: Computational overhead evaluation for system initialization.

Data size (GB)	Packing (min)		Pack compression (s)	Pack encryption (s)	Total initialization (min)
	DualPacking	LFP	Our design	Our design	Our design
2	0.33	0.60	8.43	1.13	0.56
4	2.20	2.68	16.93	2.20	2.67
6	6.08	6.60	28.69	3.24	6.85
8	11.26	11.93	61.23	4.39	12.68
10	19.57	20.12	98.61	5.57	21.75

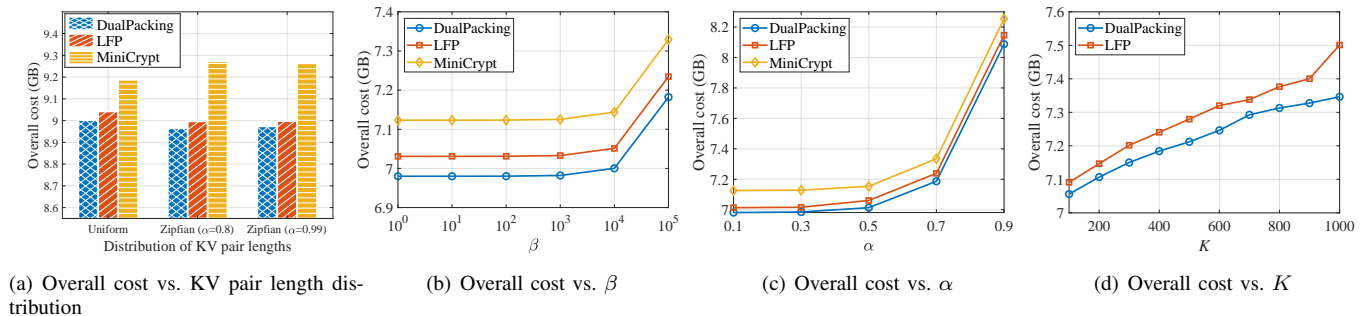


Fig. 4: Evaluation of the overall cost of data outsourcing.

an in-memory KV store, and RocksDB, a persistent SSD-based KV store. By default, we set the parameter $\beta = 10^6$ and the pack size $L^* = 0.8\text{MB}$. To facilitate evaluation, in certain experiments, we enhance DualPacking with K -indistinguishable frequency protection, setting the parameter K to 1000.

The dataset and workloads are generated using the Yahoo! Cloud Servicing Benchmark (YCSB) [20], a standard benchmark for KV stores. The dataset consists of 2.5×10^5 KV pairs, with the length distribution of KV pairs following a uniform distribution. We assess system throughput using two YCSB workloads: workload A (95% reads, 5% writes) and workload C (100% reads). YCSB uses a Zipfian distribution for key accesses. To simulate access patterns in real-world deployments [20], we set the skewness parameter α in the Zipfian distribution to 0.99, indicating a highly skewed distribution.

B. Performance Evaluation

System initialization evaluation: We first evaluate the system initialization performance of our design. During this process, the proxy utilizes the DualPacking method to organize KV pairs into packs. Each pack is then compressed, encrypted, and padded to equalize their sizes. Table II shows the latency of each main step during system initialization as the dataset size varies. The results indicate that the packing operation is the most time-consuming step. Notably, DualPacking consistently outperforms LFP in terms of packing latency. This efficiency gain occurs because, unlike LFP which relies on a computationally intensive binary search to determine the optimal number of packs, DualPacking completes the packing process by traversing all KV pairs in a single pass.

Evaluation on overall cost of data outsourcing: We then compare the performance of our design with that of LFP and MiniCrypt in terms of the overall cost of data outsourcing. We first evaluate the influence of KV pair distribution on the overall cost of data outsourcing, considering two typical types

of distribution: Zipfian and uniform. We test different values of the skewness parameter α in the Zipfian distribution to better evaluate the performance. As shown in Fig. 4(a), DualPacking outperforms LFP and MiniCrypt in all cases, which confirms the efficiency of our design. We then evaluate the impact of the weight parameter β on the overall cost of data outsourcing, as shown in Fig. 4(b). β is used to adjust the importance of bandwidth overhead in the overall cost. The results show that DualPacking consistently outperforms other designs when β varies from 1 to 10^5 .

We also evaluate the impact of varying key access distributions characterized by α . As depicted in Fig. 4(c), when α increases from 0.1 to 0.9, the overall cost of our design remains the lowest. It can also be seen that as α increases, the overall cost for all algorithms rises. This increase is because a higher α means a greater proportion of the total frequency is concentrated in the most frequently accessed KV pairs, requiring the system to send more fake queries to protect the frequency information of the KV pairs.

To further evaluate the performance of our design when frequency protection is extended to K -indistinguishable frequency smoothing, we compare the overall cost of DualPacking and LFP with varied values of K . As shown in Fig. 4(d), the overall cost of DualPacking is always lower than that of LFP when K increases from 100 to 1000. Additionally, the overall cost keeps increasing with the rise in K . The reason is that with the increase in K , more fake queries are needed to protect the frequency information.

Storage overhead evaluation: We further evaluate the performance of DualPacking in terms of storage overhead. Fig. 5(a) shows the deviation from the ideal pack number across different data sizes. The ideal pack number is determined by taking the ceiling of the quotient of the data size and the pack size. The results indicate that DualPacking consistently exhibits the smallest deviation, demonstrating its superior efficiency in managing storage compared to LFP and MiniCrypt.

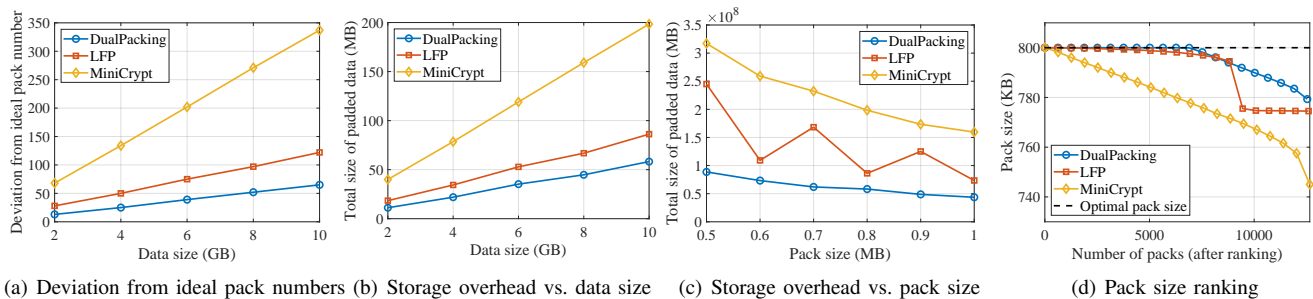


Fig. 5: Storage overhead evaluation.

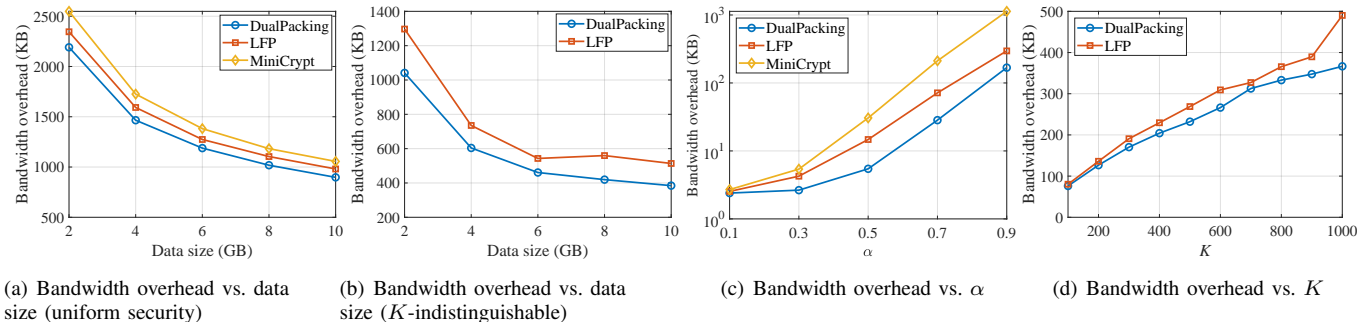


Fig. 6: Bandwidth overhead evaluation.

Fig. 5(b) illustrates the total size of padded data as the data size increases. The figure shows that DualPacking significantly reduces storage overhead. For example, when the data size reaches 10GB, DualPacking incurs only about a 50MB storage overhead in length protection, which is approximately just one-fourth of the overhead incurred by MiniCrypt.

In Fig. 5(c), we evaluate the impact of pack size on storage overhead. The results indicate that storage overhead decreases with the increase in pack size. This decrease is attributed to larger pack sizes enabling more efficient grouping of packs, resulting in less wasted space. Meanwhile, DualPacking maintains the lowest storage overhead across all pack sizes. Fig. 5(d) shows the ranking of pack sizes after packing for each design. DualPacking consistently keeps pack sizes close to the optimal value. In contrast, approximately one-third of the packs generated by LFP have sizes significantly smaller than the optimal pack size, while only a small portion of MiniCrypt’s packs approach the optimal size. This confirms the packing efficiency of our design.

Bandwidth overhead evaluation: We then analyze the bandwidth overhead of DualPacking and compare it with LFP and MiniCrypt. We first consider uniform security, where the frequency distribution over all packs is smoothed to uniform. As shown in Fig. 6(a), DualPacking achieves the lowest bandwidth overhead, followed by LFP and then MiniCrypt. Meanwhile, when the number of user requests remains the same, the bandwidth overhead decreases across all designs as the data size increases. This trend arises because the overhead is primarily influenced by the access frequency of the most frequently accessed KV pairs. Under a Zipfian distribution, the relative access frequency of the most popular KV pairs diminishes as the dataset grows, thereby reducing their contribution

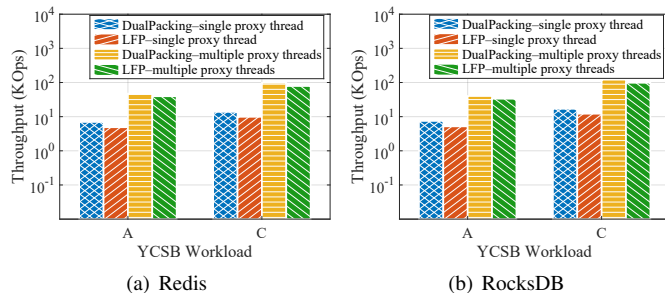


Fig. 7: Throughput evaluation.

to the overhead. We then evaluate DualPacking and LFP under K -indistinguishable frequency protection. Fig. 6(b) highlights DualPacking’s consistent superiority over LFP in this scenario. For instance, at an 8GB data size, DualPacking incurs a bandwidth overhead of 419KB, which is approximately 75% of LFP’s overhead.

We also evaluate the impact of parameters on bandwidth overhead. In Fig. 6(c), we vary the parameter α of the request access frequency distribution from 0.1 to 0.9. The results show that DualPacking consistently achieves significantly lower bandwidth overhead compared to LFP and MiniCrypt. Notably, as α increases, the bandwidth overhead for all algorithms also increases. This occurs because a higher α value implies that a larger proportion of total access frequency is attributed to high-frequency KV pairs, requiring the system to generate more fake queries to protect their frequency information. We then evaluate DualPacking and LFP under varying K values. As shown in Fig. 6(d), bandwidth overhead grows with the increase of K , yet DualPacking maintains consistent

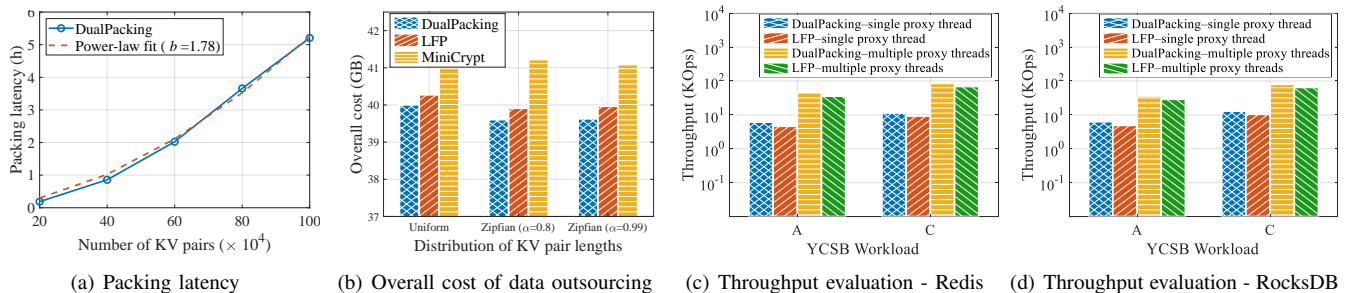


Fig. 8: System scalability evaluation for handling large-scale workloads.

superiority over LFP across all tested configurations.

Throughput evaluation: We conduct a comprehensive throughput evaluation of DualPacking. To ensure a fair comparison and to account for low throughput under uniform security conditions, we compared DualPacking and LFP with K -indistinguishable frequency protection across multiple YCSB workloads on Redis and RocksDB. Fig. 7(a) illustrates the throughput performance for single-threaded and multi-threaded operations with Redis under workloads A and C. In single-threaded mode, DualPacking achieves 7 KOps for workload A and 13.8 KOps for workload C, significantly outperforming LFP, which attains lower throughput for both workloads. For multi-threaded scenarios (8 threads), DualPacking demonstrates even greater superiority, delivering 47.4 KOps (workload A) and 97.8 KOps (workload C), while LFP lags behind. Fig. 7(b) shows results for RocksDB. The throughput trends closely align with those observed in Redis, further validating DualPacking’s high performance across different database backends.

Scalability evaluation: We further evaluate the scalability of our design for handling large-scale workloads. Specifically, we consider outsourcing a 50GB KV store with 1 million KV pairs whose lengths follow a uniform distribution and evaluate the performance of our design from three aspects: packing latency, overall cost of data outsourcing, and throughput.

In Fig. 8(a), we illustrate how the packing latency varies with the number of KV pairs. To better evaluate the scalability of DualPacking, we consider each pair of packing latency and number of KV pairs as a data point and use these points to fit a power-law function of the form $y = a \cdot n^b$, where a and b are parameters, n is the number of KV pairs, and y is the packing latency. The results show that the exponent $b = 1.78 < 2$, which is consistent with the time complexity analysis presented in Section IV-B. In terms of the overall cost of data outsourcing, the proposed DualPacking algorithm achieves the lowest cost compared to LFP and MiniCrypt, as shown in Fig. 8(b). Regarding throughput, our system consistently outperforms LFP across all evaluation scenarios, as illustrated in Fig. 8(c) and Fig. 8(d). These results demonstrate the excellent scalability of our design.

VIII. RELATED WORKS

A. Secure KV Stores

KV stores play a central role in numerous applications due to their high performance. As concerns about data se-

curity increase, substantial research has focused on secure KV stores, particularly in cloud environments [21]–[26]. One line of research adopts the technique of Intel Software Guard Extensions (SGX), which enhances data security against untrusted cloud environments by storing and processing sensitive data within a hardware-assisted trusted execution environment known as an enclave [27]–[29]. A significant limitation of SGX-based designs is their high reliance on the security of the enclave. Another line of research on secure KV stores uses cryptographic techniques to protect data [30]–[32]. In [31], Yuan *et al.* developed a distributed searchable KV store. Subsequently, considerable research has been conducted to enrich the functionalities of query operations [12], [32]. However, these methods often fail to guard against pattern-analysis attacks. Even with encrypted data, attackers can still infer sensitive information by analyzing data access patterns, such as the lengths and access frequencies of queried data [33], [34].

To obscure access patterns, numerous Oblivious RAM (ORAM) based solutions have been proposed [35]–[38]. ORAM aims to prevent attackers from deducing data content by disguising data access patterns. Despite its potential, the high storage and bandwidth overhead of ORAM limits its effectiveness [34], [39]. Recently, PANCAKE [17] has emerged as a novel solution to protect KV stores from pattern-analysis attacks. PANCAKE achieves this by smoothing the length and access frequency of all keys to uniformity, securing both length and frequency. However, the introduction of replicas in PANCAKE incurs significant storage overhead, especially for large-scale KV stores. In [18], a K -indistinguishable frequency smoothing designs is proposed to further reduce the overhead against pattern-analysis attacks.

B. Secure Data Compression

Beyond security, efficiency is another critical issue for cloud-based storage systems. Compression is widely utilized in various storage systems, such as Cassandra [40], MongoDB [41], and Redis [42], to boost performance. It not only reduces storage overhead but also enhances overall system efficiency [2]. However, while both encryption and compression are important for KV stores, their integration has been limited due to their inherent incompatibility [12].

MiniCrypt [11] is the first design that combines compression and encryption for KV stores without compromising efficiency. It divides KV pairs into packs, which are then compressed

and encrypted at the pack level. Subsequently, TinyEnc [12] was proposed to enhance this approach by providing rich query supports. However, due to variations in access frequency and the sizes of packs returned by the cloud in response to user queries, these designs are vulnerable to pattern-analysis attacks. For this problem, Zhang *et al.* [19] proposed a secure and compressed KV storage system with pattern-analysis security, featuring a length-first packing algorithm to minimize the difference in pack lengths. However, this algorithm overlooks the overhead associated with frequency information protection, potentially leading to significant bandwidth overhead. Therefore, there is a pressing need to design a packing algorithm that considers both length and frequency protection, which is the focus of this paper.

IX. CONCLUSION

In this paper, we explore the optimal packing problem for encrypted and compressed KV stores, with the goal of minimizing the overhead of protection against pattern-analysis attacks. We first formally formulate the cost-efficient and pattern-protected packing problem and then develop DualPacking, a two-dimensional packing algorithm. A detailed analysis of DualPacking is provided to demonstrate its high efficiency. Subsequently, we develop an encrypted and compressed KV storage system that can effectively handle dynamic changes within KV stores. Finally, we provide a comprehensive security analysis and conduct extensive experiments to validate the efficiency of DualPacking. The experimental results confirm that the proposed design can protect against pattern-analysis attacks while minimizing the overall cost of data outsourcing.

REFERENCES

- [1] A. Khandelwal, R. Agarwal, and I. Stoica, "Blowfish: Dynamic storage-performance tradeoff in data stores," in *Proc. 13th USENIX Symp. Netw. Syst. Des. Implementation*, 2016.
- [2] R. Agarwal, A. Khandelwal, and I. Stoica, "Succinct: Enabling queries on compressed data," in *Proc. 12th USENIX Symp. Netw. Syst. Des. Implementation*, 2015, pp. 337–350.
- [3] X. Yuan, X. Wang, C. Wang, B. Li, and X. Jia, "Enabling encrypted rich queries in distributed key-value stores," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 6, pp. 1283–1297, 2018.
- [4] Ciphercloud, Online at <http://www.ciphercloud.com/>.
- [5] Navajo Systems, Online at <http://tinyurl.com/y85obds6>.
- [6] C. Zhang, Y. Miao, Q. Xie, Y. Guo, H. Du, and X. Jia, "Privacy-preserving deduplication of sensor compressed data in distributed fog computing," *IEEE Trans. Parallel Distributed Syst.*, vol. 33, no. 12, pp. 4176–4191, 2022.
- [7] Q. Xie, C. Zhang, and X. Jia, "Security-aware and efficient data deduplication for edge-assisted cloud storage systems," *IEEE Trans. Services Comput.*, vol. 16, no. 3, pp. 2191–2202, 2023.
- [8] Y. Guo, C. Zhang, C. Wang, and X. Jia, "Towards public verifiable and forward-privacy encrypted search by using blockchain," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 3, pp. 2111–2126, 2023.
- [9] S. Huang, J. Xie, and M. M. A. Muslam, "A cloud computing based deep compression framework for uhd video delivery," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1562–1574, 2022.
- [10] R. Xie and X. Jia, "Transmission-efficient clustering method for wireless sensor networks using compressive sensing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 806–815, 2013.
- [11] W. Zheng, F. Li, R. A. Popa, I. Stoica, and R. Agarwal, "Minicrypt: Reconciling encryption and compression for big data stores," in *Proc. 12th Eur. Conf. Comput. Syst.*, 2017, pp. 191–204.
- [12] S. Qi, J. Wang, M. Miao, M. Zhang, and X. Chen, "Tinyenc: Enabling compressed and encrypted big data stores with rich query support," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 1, pp. 176–192, 2023.
- [13] M. Zhang, S. Qi, M. Miao, and F. Zhang, "Enabling compressed encryption for cloud based big data stores," in *Proc. Cryptol. Netw. Secur. (CANs)*. Berlin, Germany: Springer-Verlag, 2019, pp. 270–287.
- [14] C. Zhang, Q. Xie, M. Wang, Y. Guo, and X. Jia, "Optimal compression for encrypted key-value store in cloud systems," *IEEE Trans. Comput.*, vol. 73, no. 3, pp. 928–941, 2024.
- [15] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *Proc. NDSS Symp.*, 2012.
- [16] S. Lambregts, H. Chen, J. Ning, and K. Liang, "Val: Leakage-abuse attack with leaked documents," in *Proc. Eur. Symp. Res. Comput. Secur. Cham*: Springer, 2022, pp. 653–676.
- [17] P. Grubbs, A. Khandelwal, M.-S. Lacharité, L. Brown, L. Li, R. Agarwal, and T. Ristenpart, "Pancake: Frequency smoothing for encrypted data stores," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 2451–2468.
- [18] C. Zhang, Q. Xie, Y. Miao, and X. Jia, "K-indistinguishable data access for encrypted key-value stores," in *Proc. IEEE 42nd Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2022, pp. 1144–1154.
- [19] C. Zhang, Y. Ming, M. Wang, Y. Guo, and X. Jia, "Encrypted and compressed key-value store with pattern-analysis security in cloud systems," *IEEE Trans. Inf. Forensics Secur.*, vol. 19, pp. 221–234, 2024.
- [20] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proc. 1st ACM Symp. Cloud Comput.*, 2010, pp. 143–154.
- [21] C. Zhang, Y. Guo, X. Jia, C. Wang, and H. Du, "Enabling proxy-free privacy-preserving and federated crowdsourcing by using blockchain," *IEEE Internet Things J.*, vol. 8, no. 8, pp. 6624–6636, 2021.
- [22] X. Yuan, Y. Guo, X. Wang, C. Wang, B. Li, and X. Jia, "Enckv: An encrypted key-value store with rich queries," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 423–435.
- [23] Y. Hu, X. Yao, R. Zhang, and Y. Zhang, "Freshness authentication for outsourced multi-version key-value stores," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 3, pp. 2071–2084, 2023.
- [24] M. Campanelli, F. Engelmann, and C. Orlandi, "Zero-knowledge for homomorphic key-value commitments with applications to privacy-preserving ledgers," in *Int. Conf. Secur. Cryptogr. Netw.* Springer, 2022, pp. 761–784.
- [25] Z. Jiang, X. Guo, T. Yu, H. Zhou, J. Wen, and Z. Wu, "Private set intersection based on lightweight oblivious key-value storage structure," *Symmetry*, vol. 15, no. 11, p. 2083, 2023.
- [26] H. Sun, G. Chen, Y. Yue, and X. Qin, "Improving lsm-tree based key-value stores with fine-grained compaction mechanism," *IEEE Trans. Cloud Comput.*, vol. 11, no. 4, pp. 3778–3796, 2023.
- [27] F. Yang, Y. Chen, Y. Lu, Q. Wang, and J. Shu, "Aria: Tolerating skewed workloads in secure in-memory key-value stores," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, 2021, pp. 1020–1031.
- [28] T. Kim, J. Park, J. Woo, S. Jeon, and J. Huh, "Shieldstore: Shielded in-memory key-value storage with sgx," in *Proc. 14th EuroSys Conf.*, Dresden, Germany, Mar. 2019, pp. 1–15.
- [29] I. Messadi, S. Neumann, N. Weichbrodt, L. Almstedt, M. Mahhoudi, and R. Kapitza, "Precursor: A fast, client-centric and trusted key-value store using rdma and intel sgx," in *Middleware*, 2021, pp. 1–13.
- [30] X. Yuan, C. Cai, C. Wang, and Q. Wang, "A scalable ledger-assisted architecture for secure query processing over distributed iot data," *CCF Trans. Netw.*, vol. 3, no. 2, pp. 97–111, 2020.
- [31] X. Yuan, X. Wang, C. Wang, C. Qian, and J. Lin, "Building an encrypted, distributed, and searchable key-value store," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, May 2016, pp. 547–558.
- [32] W. Lin, H. Cui, B. Li, and C. Wang, "Privacy-preserving similarity search with efficient updates in distributed key-value stores," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1072–1084, 2021.
- [33] S. Lambregts, H. Chen, J. Ning, and K. Liang, "Volume and access pattern leakage-abuse attack with leaked documents," *Cryptology ePrint Archive*, 2022.
- [34] Y. Dou and H. C. B. Chan, "Leakage-suppressed encrypted keyword queries over multiple cloud servers," *IEEE Trans. Cloud Comput.*, vol. 12, no. 1, pp. 26–39, 2024.
- [35] H. Liu, X. Lu, S. Duan, Y. Zhang, and Y. Xiang, "An efficient oblivious random data access scheme in cloud computing," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1940–1953, 2023.
- [36] H. Ma and X. Wang, "Cloud storage audit scheme based on oram and sgx," in *Proc. Int. Conf. Intell. Informat. Biomed. Sci. (ICHIBMS)*, vol. 8, 2023, pp. 352–356.
- [37] K. G. Larsen and J. B. Nielsen, "Yes, there is an oblivious ram lower bound!" in *Proc. Annu. Int. Cryptol. Conf.* Springer, 2018, pp. 523–542.

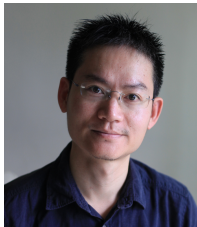
- [38] J. G. Chamani, D. Papadopoulos, M. Karbasforushan, and I. Demertzis, "Dynamic searchable encryption with optimal search in the presence of deletions," in *Proc. 31th USENIX Secur. Symp.*, 2022, pp. 2425–2442.
- [39] C. Huang, D. Liu, A. Yang, R. Lu, and X. Shen, "Multi-client secure and efficient dpf-based keyword search for cloud storage," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 1, pp. 353–371, 2024.
- [40] Cassandra, "Cassandra," <https://cassandra.apache.org/>, August 2024.
- [41] MongoDB, "Mongodb," <https://www.mongodb.com/>, August 2024.
- [42] Redis, "Redis," <https://redis.com/>, August 2024.



Chen Zhang is currently an assistant professor at the Department of Computer Science, The Hang Seng University of Hong Kong. She received her B.E. degree in Network Engineering from Harbin University of Science and Technology in 2017, the M.E. degree in Computer Science and Technology from Harbin Institute of Technology, Shenzhen in 2019, and Ph.D. degree in Computer Science from City University of Hong Kong in 2023. Her research interests include cloud computing security, mobile edge computing, federated learning, and blockchain.



Shujin Ye received the BE degree in computer science and technology from South China Normal University, the MS degree in software engineering from South China University of Technology, and the PhD degree in computer science from Hong Kong Baptist University. His research interests include cloud computing and swarm intelligence.



Hai Liu is an Professor with Department of Computer Science, The Hang Seng University of Hong Kong (HSUHK). Before joining HSUHK, he held several academic posts at University of Ottawa and Hong Kong Baptist University. Dr Liu received PhD in Computer Science at City University of Hong Kong, and received MSc and BSc in Applied Mathematics at South China University of Technology. His research interest includes wireless networking, cloud computing, artificial intelligence and algorithm design and analysis.



Tse-Tin Chan received his B.Eng. (First Class Hons.) and Ph.D. degrees in Information Engineering from The Chinese University of Hong Kong (CUHK), Hong Kong SAR, China, in 2014 and 2020, respectively. He is currently an Assistant Professor with the Department of Mathematics and Information Technology, The Education University of Hong Kong (EdUHK), Hong Kong SAR, China. Prior to this, he was an Assistant Professor with the Department of Computer Science, The Hang Seng University of Hong Kong (HSUHK), Hong Kong SAR, China, from 2020 to 2022. His research interests include wireless communications and networking, Internet of Things (IoT), age of information (AoI), and AI in wireless communications.