IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 22 April 2024, accepted 4 May 2024, date of publication 17 May 2024, date of current version 24 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3402544

RESEARCH ARTICLE

DEDUCT: A Secure Deduplication of Textual Data in Cloud Environments

KIANA GHASSABI AND PEYMAN PAHLEVANI
Department of Computer Science, Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan 4515768315, Iran
Corresponding author: Kiana Ghassabi (Kianaghassabi@iasbs.ac.ir)

ABSTRACT The exponential growth of textual data in Vision-and-Language Navigation tasks poses significant challenges for data management in large-scale storage systems. Data deduplication has emerged as a practical strategy for data reduction in large-scale storage systems; however, it has also raised security concerns. This paper introduces DEDUCT, an innovative data deduplication method for textual data. DEDUCT employs a hybrid approach that combines cloud-side and client-side deduplication mechanisms to achieve high compression rates while maintaining data security. DEDUCT's lightweight preprocessing and client-side deduplication make it suitable for resource-constrained devices like IoT devices. It has also been designed to resist side-channel attacks. Experimental evaluations on the Touchdown dataset, consisting of human-written navigation instructions for routes, demonstrate the effectiveness of DEDUCT. It achieves compression rates of nearly 66%, significantly reducing storage requirements while preserving the confidentiality of textual data. This substantial reduction in storage demands can lead to significant cost savings and improved efficiency in large-scale data management systems.

INDEX TERMS Cloud service provider, compression, secure data deduplication, textual data deduplication.

I. INTRODUCTION

Vision-and-Language Navigation (VLN) [1] tasks are becoming increasingly important due to their significant impact on advancing autonomous vehicles and intelligent systems. VLN technology empowers agents to navigate real-world environments, enhancing human-robot interactions and safeguarding safety in autonomous vehicle operations. Beyond navigation, VLN applications extend to diverse domains, including robotics, virtual assistants, and smart homes, making human-machine interactions more intuitive and user-friendly. The significance of textual data in VLN cannot be overstated, as it is the foundation for communication between humans and autonomous agents. Users convey detailed navigational commands through natural-language instructions, and autonomous systems rely heavily on the accurate interpretation and execution of these textual directives. Efficient data management has become critical to meet the increasing demands of VLN and its associated applications.

The associate editor coordinating the review of this manuscript and approving it for publication was Jie Gao.

Data deduplication [2] is a highly effective technique for reducing storage space consumption by eliminating the need for storing identical files or data blocks multiple times. Instead, only one copy of each unique data is stored, and references are used to point to the original copy. This method is particularly beneficial in cloud environments where vast amounts of data are typically stored. In backup applications, deduplication can reduce storage needs by up to $90-95\%$ [5], while in standard file systems, it can lead to a reduction of up to 68% [4].

There are three main categories of data deduplication techniques based on granularity [53]: file level, fixed-size block, and variable-sized block. File-level deduplication finds and removes entire duplicate files. Fixed-size block deduplication divides a file into fixed-size blocks and eliminates duplicate blocks. Variable-sized block deduplication utilizes various sizes of chunks to identify redundant data, but it may create more metadata and lead to hash collisions. Block-level deduplication is typically more efficient as it can detect duplicates even if they are stored across different files or portions of the storage system. Deduplication techniques can also be categorized based on place: server-based and

client-based. Server-based deduplication identifies and eliminates duplicate data on the server. Server-based deduplication eliminates the need for users to perform deduplication tasks locally. However, server-side deduplication may only partially mitigate communication overhead. On the other hand, client-side deduplication takes place on the user's device before uploading data to the cloud. It involves collaboration between the client and server to find redundant data. This can significantly reduce bandwidth consumption by sending only unique data. However, client-side deduplication raises concerns regarding side-channel attacks [37] and data leakage. Finally, deduplication can be classified based on time: inline and offline. Inline deduplication eliminates duplicate data before or as it is being stored. Offline deduplication deals with deduplication after data is stored on a storage device.

Classic Deduplication (CD) methods [9] primarily focus on identifying and removing duplicate files, which can lead to inefficient storage when files share similar content but are not identical. Generalized Deduplication (GD) [20] has emerged as a more comprehensive approach to address this limitation. GD expands the scope of traditional methods by recognizing and eliminating nearly identical or similar data chunks. This reduces storage requirements significantly, eliminates data redundancy, and improves data management efficiency.

Textual data deduplication, while receiving less attention than other data types, has become increasingly crucial due to the exponential growth in textual information generation. There may be more efficient approaches than cloud-based solutions, and clients can significantly contribute by performing initial data preprocessing. Nevertheless, existing client-side deduplication methods face security challenges, particularly in cross-user deduplication scenarios. The risk of side-channel attacks, where unauthorized access to files uploaded by other users is possible, underscores the need for an enhanced system model for textual data deduplication. Our motivation stems from addressing the limitations of current client-side deduplication approaches.

A. CONTRIBUTION

We propose DEDUCT (DEDUplication for Cloud Text), a deduplication method explicitly designed for textual data. DEDUCT builds upon the framework presented in [8], emphasizing data security and optimization of storage efficiency. It employs a hybrid approach integrating cloud-based deduplication with lightweight preprocessing tasks on resource-constrained clients. This hybrid approach optimizes data storage, enhances performance, and safeguards data confidentiality in various applications, including VLN tasks and other domains. Our contributions can be summarized as follows:

- **Novel Client-Side Deduplication:** DEDUCT eliminates the need to signal between the client and the cloud, minimizes communication overhead, prevents data leakage, and mitigates side-channel attacks. Additionally, it minimizes duplicate transmissions in the channel without requiring cloud involvement.
- **Data Confidentiality Preservation:** DEDUCT ensures data confidentiality through client-side preprocessing, where data remains encrypted and secure throughout deduplication. This client-based approach is adaptable to resource-constrained devices, such as IoT, mobile, embedded systems, and edge computing devices, making it applicable in various scenarios.
- **Enhanced Load Balancing:** DEDUCT distributes deduplication tasks between the cloud and the client, mitigating the processing load on the cloud server and improving overall storage system performance.

Our experimental results illustrate that, in the best-case scenario, DEDUCT achieves a lossless compression ratio of nearly 66% on the Touchdown [13] dataset, highlighting its effectiveness in reducing storage costs.

The rest of the paper is organized as follows: Section II covers preliminaries and fundamental concepts. Section III defines the system and adversary models that form the foundation for our deduplication framework. Our proposed scheme is detailed in Section IV, and performance metrics are discussed in Section V. In Section VI, we assess the security of our proposed method. Section VII summarizes the strategies used to prevent the information loss in the context of textual data deduplication. Section VIII presents the evaluation results and Section IX reviews related works on generalized deduplication and privacy in deduplication systems. Finally, Section X concludes the paper by summarizing the key contributions of DEDUCT and outlining future research directions.

II. PRELIMINARIES

This section introduces the fundamental definitions, notations, and methodologies for data deduplication and the proposed method.

A. FINGERPRINTS

In data deduplication, fingerprints serve as unique identifiers for specific chunks or blocks of data. The fingerprints are generated using various hash algorithms like MD5 [21], SHA-1 [22], and SHA-256 [23]. Alternatively, the Cyclic Redundancy Check (CRC) [24] algorithm can also generate fingerprints for data blocks. While algorithms like SHA-1 are generally considered more robust for encryption and authentication purposes compared to CRC, the choice of fingerprinting algorithm depends on the application's specific requirements. Factors like collision probability, strength, performance, and length must be considered when selecting an appropriate algorithm.

More recent works explore verifiable array authenticated data structures (VADS) [52]. VADS play a crucial role in ensuring the integrity and authenticity of data. These structures, exemplified by the verifiable Shrubs array (VSA), offer advantages over traditional Merkle tree-based

structures. Unlike Merkle trees, VSAs facilitate append-only writes of digests, eliminating redundant hash computations and random I/O operations. Furthermore, authenticated data within VSAs can be efficiently stored and verified from a file, simplifying the storage and retrieval process. Incorporating such VADS enhances the robustness and reliability of systems, making them more resilient to data tampering and ensuring higher levels of trust in data integrity.

B. TOKENIZATION

Tokenization [27] is a common technique used in Natural Language Processing (NLP) to split textual data into smaller units called ''tokens''. Tokens can be words, numbers, or even individual symbols. The primary aim of tokenization is to divide the text into smaller, meaningful units that can be further analyzed and processed.

Tokenization can be performed using different methods, such as whitespace-based tokenization, rule-based tokenization, and statistical tokenization [28]. Whitespace or punctuation is often used as delimiters to separate words in whitespace-based tokenization. This approach assumes that spaces or specific characters separate words. Rule-based tokenization, on the other hand, relies on predefined rules or patterns to identify and extract tokens from the text. These rules can be based on language-specific grammatical structures or syntactic patterns. Statistical tokenization utilizes probabilistic models to determine token boundaries based on statistical properties of the text, such as word frequencies or sequence patterns. The choice of tokenization method depends on the textual data's characteristics and the system's specific requirements.

C. LEMMATIZATION

Lemmatization [42] is a commonly used technique in NLP to identify a word's dictionary or base form. Unlike stemming [42], which removes suffixes to reduce words to their root form, lemmatization considers the word's context and morphological analysis to derive its base form accurately. By converting words to their lemmas, which are canonical representations, lemmatization helps improve the accuracy and effectiveness of text analysis and information retrieval tasks. Lemmatization is particularly useful when word meaning and context are crucial, such as in language understanding, information extraction, and text classification.

D. LEVENSHTEIN DISTANCE AND WAGNER-FISCHER ALGORITHM

The Levenshtein distance [29], also known as edit distance, is a metric that measures the difference between two strings based on the minimum number of single-character edits required to transform one string into the other. The edits include insertions, deletions, or substitutions of characters. Formally, let $A \stackrel{\text{def}}{=} (a_1, \ldots, a_n)$ be a string over alphabet Σ. We define the possible editing operations on A:

1) Delete the i-th position to get $(a_1, ..., a_{i-1}, a_{i+1}, ..., a_n)$.
2) Insert $b \in \Sigma$ at position $(i+1)$ to obtain $(a_1, ..., a_i, b, a_{i+1}, ..., a_n)$.
3) Change the position i to $b \in \Sigma$ to obtain $(a_1, ..., a_{i-1}, b, a_{i+1}, ..., a_n)$.

The Wagner-Fischer algorithm [30] is a dynamic programming approach for comparing two sequences. The algorithm constructs a matrix with several rows and columns equivalent to the lengths of the two strings being compared. The rows represent the first sequence, while the columns represent the second. The algorithm cumulatively computes all possible alignments between two sequences by finding the path through the matrix that minimizes the penalties induced by gaps and mismatching segments. The final edit distance is obtained by resolving all the edit distances of the substrings that constitute the final strings. The key idea is to solve all sub-problems using values obtained from previous computations. Formally, let $A \stackrel{\text{def}}{=} (a_1, \ldots, a_n)$ and $B \stackrel{\text{def}}{=} (b_1, \ldots, b_m)$ be two strings in Σ^*. For $i \in [n]$ the set $\{1, 2, \ldots, n\}$, denote the sub-string $A^{(i)} \stackrel{\text{def}}{=} (a_1, a_2 \ldots, a_i)$ and the edit distance between $A^{(i)}$ and $B^{(i)}$ by $D(i, j)$. The goal is thus to find $D(n, m)$. Algorithm 1 shows the steps.

Algorithm 1 Wagner-Fischer Algorithm

input : Two chains $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$.
output: An integer value with the edit distance between the chains A and B.

Let t be an $n \times m$ matrix with indexes starting from 1.
for $(i, j) \in [n] \times [m]$ **do**
  if $a_i \neq b_j$ **then**
    $t(i, j) = 1$
  else
    $t(i, j) = 0$
Let D be an $(n+1) \times (m+1)$ zero-initialized matrix, indexes starting from 0.
for $i = 0$ *to* n **do**
  $D(i, 0) = i$
for $j = 0$ *to* m **do**
  $D(0, j) = j$
for $i = 1$ *to* n **do**
  for $j = 1$ *to* m **do**

$$D(i,j) = min \begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i-1,j-1)+t(i,j) \end{cases}$$

return $D(n, m)$

E. GENERALIZED DEDUPLICATION

Traditional data deduplication methods solely focus on identifying exact duplicates. However, this approach may not be optimal for scenarios where data chunks share significant similarities but are not identical. Generalized Deduplication

(GD) [20] has emerged as a more comprehensive technique to address this limitation. This approach includes a transformation step, transforming each data chunk into a base and a deviation. The objective is to recognize similarities within the data, such as chunks sharing the same base, to achieve greater compression potential. The base value is pivotal in deduplication, while the deviation value captures the differences between the original data chunk and the extracted base. GD utilizes transformation functions such as Hamming [14] or Reed-Solomon codes [20] to optimize storage costs. The overall processes of CD and GD are compared in Fig.1 and Fig.2. In this example, the data is split into 4 chunks ch_1, ch_2, ch_3, and ch_4. The chunks ch_2 and ch_3 are identical, and ch_1 and ch_4 are similar. The CD does not distinguish similar data; only the identical duplicate data will be identified and assigned a pointer. GD, however, can provide this by using the transformation function. Similar bases are deduplicated, and deviations are assigned to them.

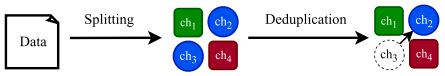


FIGURE 1. The overall process of classic deduplication.

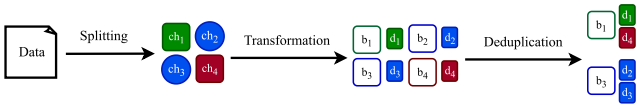


FIGURE 2. The overall process of generalized deduplication.

III. MODELS

In this section, we describe the system and adversary models of the proposed method.

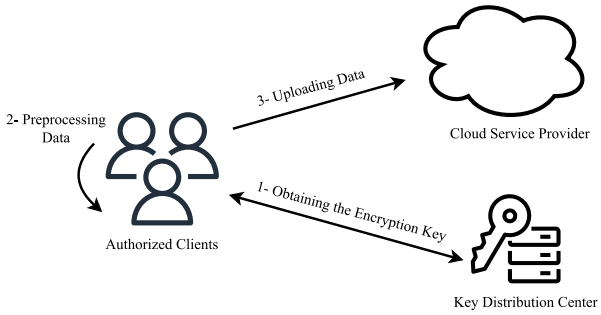


FIGURE 3. The system model of DEDUCT.

A. SYSTEM MODEL

As shown in Fig.3, the DEDUCT system model comprises three components as follows:

- **Key Distribution Center (KDC)**: The KDC serves as a central authority responsible for distributing encryption keys to authorized clients. To obtain an encryption key, a client sends its unique group ID (ID_{Client}) to the KDC. The KDC verifies the client's identity and authenticity using a secure authentication protocol (e.g., challenge-response or ticketing schemes). If the client is authenticated, the KDC generates a unique encryption key for the client and securely transmits it to the client's device. The KDC is no longer required once the system setup phase is complete.
- **Cloud Service Provider (CSP)**: The CSP stores encrypted data uploaded by clients. It utilizes a pointer-based approach to efficiently manage storage space and mitigate duplicate data.
- **Authorized Clients**: Clients are users who belong to specific groups or organizations and have access to the KDC for key retrieval. Before the initial data transmission, clients communicate with the KDC to obtain the key for encrypting specific data segments (bases). Clients perform a five-step process before uploading data to the CSP. First, data is divided into smaller tokens using a tokenization algorithm. Then, each token is transformed into a base and deviation pair by employing the Wagner-Fischer algorithm. Next, the client generates a unique identifier for each base by calculating its CRC value, which is stored locally for future reference. The base is then encrypted using the obtained encryption key and a chosen encryption algorithm to preserve confidentiality and integrity. Finally, the client uploads the encrypted base, corresponding CRC value, and deviation to the CSP. Only the CRC value and deviation are transmitted if the base's CRC value already exists locally. We also assume that clients have limited storage space, so the system is designed to work within this constraint. Moreover, integrating advanced cryptographic techniques such as Verifiable Authenticated Data Structures (VADS) for enhanced data integrity and audibility is left as future work. The details of each step are explained in Section IV.

The core of DEDUCT's deduplication mechanism lies in its combination of tokenization, transformation, CRC computing, and pointer-based storage. Tokenization breaks down large data blocks into smaller, more manageable tokens, reducing the computational overhead associated with deduplication operations. The transformation step converts tokens into base and deviation pairs using the Wagner-Fischer algorithm, enabling more precise identification of duplicate data. CRC calculation generates unique identifiers for each base, facilitating efficient comparison and deduplication. Local CRC storage allows clients to avoid unnecessary transmission of CRC values to the cloud for duplicate data. By performing CRC computations locally, clients minimize bandwidth consumption and also reduce the burden on the cloud infrastructure. Finally, pointer-based storage at the cloud side eliminates the need to store identical encrypted data blocks by maintaining pointers to existing values, significantly reducing storage requirements.

TABLE 1. Notations used in the proposed scheme.

Notation	Description
D	Data
t	Token
T	A set of tokens
b	Base
B	A set of bases
d	Deviation
c	CRC value of the base
τ	Threshold
k	Encryption Key
e	Encrypted value of the base
ptr	Pointer
$\mathrm{ID}_{\mathrm{Client}}$	An ID of a client who belongs to a group
TGT	A Ticket-Granting Ticket

B. ADVERSARY MODEL

We assume that the CSP is honest but curious, i.e., it accurately executes system operations but may attempt to examine data content. Additionally, we do not consider scenarios involving collusion between users and the CSP. The KDC is also assumed to be entirely trustworthy and resistant to compromise by any adversary [39]. The following summarizes our adversary model:

1) MALICIOUS CSP

The CSP can execute all user interactions following designated protocols. However, the CSP can launch offline brute-force attacks. By systematically generating candidate data chunks and computing their corresponding CRC values, the CSP could compare them with the received CRCs to identify potential matches.

2) MALICIOUS CLIENT

We assume the malicious client is a legitimate user accessing the KDC to obtain authentic encryption keys. In addition, the CSP is always honest when performing all system protocols. A malicious client can exploit vulnerabilities to launch several attacks. One such attack is the Poisoning attack [47], where the malicious client replaces valid encrypted data with a tampered version. This modification makes it challenging for the cloud server to validate the authenticity of the original data, potentially resulting in users retrieving a corrupted version instead of their original files. Another potential attack in client-side deduplication is the online brute-force attack, which arises in scenarios where the cloud directly provides feedback regarding the existence of specific data.

IV. PROPOSED SCHEME

This section explores the proposed scheme, DEDUCT, a novel approach to secure and efficient textual data deduplication in cloud storage. It outlines the key steps in the client-side and cloud-side processes, as presented in Algorithm 2. To ensure clarity, Table 1 explains the notations utilized throughout this section.

A. CLIENT-SIDE

As mentioned before, the client-side process comprises five steps: Obtaining the Encryption Key, Data Splitting, Transformation, Encryption, and CRC Computing. The entire process is shown in Fig.4. The following sections describe each part in more detail.

Algorithm 2 DEDUCT

input : D: Raw Data,
τ: Threshold.

```
// Client-Side: Obtaining Encryption Key from the KDC
k ← ObtainKey(ID_Client);
// Client-Side: Splitting
T ← Tokenize(D);
// Client-Side: Transformation
for each t ∈ T do
    // Client-Side: Extract base and deviation
    b, d = Transform(t);
    // Client-Side: CRC Computing
    c ← CRC(b);
    if c ∈ localStorage then
        SendtoCloud(c, d);
    else
        // Client-Side: Encryption
        e ← Encrypt(b, k);
        if Size(c) ≤ AvailableStorage(localStorage) then
            localStorage.append(c);
            SendtoCloud(c, e, d);
        else
            // Remove the one with the least frequency
            min ← MinFreq(localStorage);
            localStorage.remove(min);
            localStorage.append(c);
            SendtoCloud(c, e, d);
// Cloud-Side
for each Recieved Message do
    if Recieved Message == (c, d) then
        e' ← RetrieveEncryptedData(c);
        ptr ← createPointer(e');
        Store(d, ptr);
    else if Recieved Message == (c, e, d) then
        e' ← RetrieveEncryptedData(c);
        if e = e' then
            ptr ← createPointer(e');
            Store(d, ptr);
        else
            cloudStorage.append(c);
            Store(d, e);
    else
        cloudStorage.append(c);
        Store(d, e);
```

1) OBTAINING THE ENCRYPTION KEY

To initiate the data encryption, clients must first obtain the encryption key (k) from the KDC. Since clients are assigned to specific groups or organizations, the KDC must ensure that all authorized users within the same group receive the same encryption key. To verify the client's authenticity, the KDC employs Kerberos authentication [50], which is a ticket-based authentication mechanism. During the authentication process, clients register with the KDC and provide their unique group ID (ID_{Client}). The KDC then verifies the client's credentials and grants a Ticket-Granting Ticket (TGT), an encrypted ticket containing authentication information and a session key. The client uses this session key to decrypt the TGT and extract the encryption key specific to its group. To further protect the encryption key from unauthorized access and tampering, the KDC establishes a secure Transport Layer Security (TLS) connection with the client before transmitting the key. This secure communication channel guarantees the confidentiality and integrity of the key throughout transmission. Algorithm 2 outlines the process of obtaining k using the *ObtainKey*(.) function.

2) DATA SPLITTING

During this step, the client uses tokenization to divide the data into smaller pieces.

***Definition** 1:* Let D be the original data, and $T = t_1, t_2, \ldots, t_n$ be the set of tokens resulting from the tokenization algorithm. The tokenization algorithm, denoted as Tokenize(D), partitions D into a sequence of non-overlapping tokens such that T represents the set of all generated tokens.

The algorithm ensures that the resulting tokens cover the entire data, enabling subsequent processing steps to operate on manageable units. As illustrated in Fig.4, the data is divided into multiple tokens denoted as $t_1, t_2, \ldots,$ and t_n.

While tokenization itself is not a security measure, it can be a crucial preprocessing step for enhancing data security. By breaking down data into smaller tokens, tokenization reduces the potential impact of a security breach. Even if an attacker gains access to tokens, they may not be able to reconstruct the original data without additional information.

3) TRANSFORMATION

The transformation step is a critical phase in DEDUCT, involving the conversion of each token into a base and deviation, as defined below:

***Definition** 2:* Let D denote the original data, and $T = t_1, t_2, \ldots, t_n$ represent the set of tokens resulting from the tokenization algorithm. The Transformation process, denoted as Transform(t_i), converts the input token into a corresponding base-deviation pair (b_i, d_i). For every token t_i in T, the transformation involves applying a method, such as the Wagner-Fischer algorithm, to derive a base and deviation.

Algorithm 3 Transformation

```
input : T: a set of tokens (t_1, t_2, ..., t_n),
        τ: Threshold.
for each t ∈ T do
    // Extracting the base of a token
    b ← Lemmatize(t);
    // computing the edit distance
    distance ← LevenshteinDistance(b, t);
    // Computing the deviation
    if distance < τ then
        d ← WagnerFischer(b, t);
    else
        // Use token as its own base
        b ← t;
        d ← 0;
    return b, d
```

Algorithm 3 outlines the step-by-step execution of the transformation function. For each token t_i in T, the process begins by extracting the base of the token using lemmatization. Subsequently, the edit distance between the base and the token is computed using the Levenshtein distance algorithm.

The deviation represents the differences between a token and its base. To limit the number of allowed operations for this conversion, a threshold value τ is used. If the computed distance is less than a specified threshold τ, the deviation is determined using the WagnerFischer algorithm. However, if the distance exceeds τ, the token is considered identical to its base, and both b and d are set accordingly ($b \leftarrow t; d \leftarrow 0$).

The deviation consists of three components, as illustrated in Fig.5:

- Operation type (o_t) specifies the editing operation performed on the base to obtain the token. The possible operation types are insertion (I), deletion (D), or substitution (S).
- Base index (o_i) indicates the position in the base where the operation o_t is applied.
- Value (o_v) represents the data inserted, deleted, or substituted during the operation o_t.

We will now provide a formal definition for a deviation. The transformation of a base (b) to its corresponding token (t) can be expressed as a set of operations (op). This set can be denoted as $O_{b\to[t]} = [\mathrm{op}_1, \mathrm{op}_2, \ldots, \mathrm{op}_n]$.

Several operations can be used to convert a base to its corresponding token. These operations are represented by $P_{b\to t}$ and are defined as follows:

$$P_{b\to t} = [O^1_{b\to t}, O^2_{b\to t}, \ldots, O^m_{b\to t}]. \quad (1)$$

To determine the deviation between b and t, we need to find the operation set from $P_{b\to t}$ that requires the least number of operations and stays within the defined τ. Therefore, we can

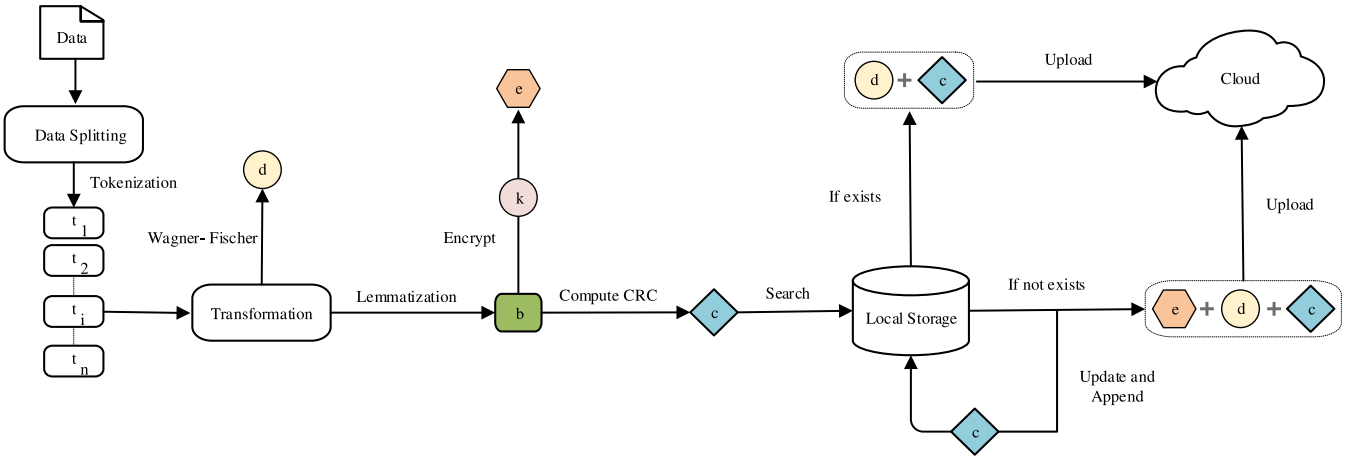


FIGURE 4. The overall process of DEDUCT on the client side.

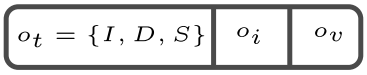


FIGURE 5. The structure of an operation (op): the operation type (o_t), the base index to which the operation applies (o_i), and the value involved in the operation (o_v).

define the deviation for a pair of b and t as follows:

$$\text{dev}_{b\to t} = \Big(O_{b\to t} \mid \forall(O^i_{b\to t}) \in P_{b\to t}, \; |O_{b\to t}| \le |O^i_{b\to t}| \text{ and } |O_{b\to t}| < \tau\Big). \quad (2)$$

The example in Fig.6 illustrates the lemmatization of the word "studies" to its base value "study" using a threshold value of $\tau = 4$. There are two potential sequences of operations, or operation sets, that can transform the base into its corresponding token. The first set ($O^1_{b\to t}$) involves three operations (op_1, op_2, and op_3), while the second set ($O^2_{b\to t}$) comprises only two operations (op_1 and op_2). For the first set, in op_1, the substitution operation (S) replaces the character "y" with "i" at the fourth index. To determine the deviation, we select the operation set that meets the transformation criteria with the minimum number of operations, staying within the threshold. In this case, the second set ($O^2_{b\to t}$) is selected as it satisfies the transformation criteria with the least number of operations within the threshold. Additionally, it is important to note that the deviation is kept unchanged throughout the process.

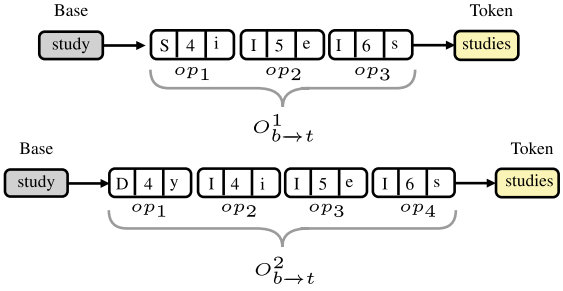


FIGURE 6. An example of determining the deviation in DEDUCT.

4) CRC COMPUTING

In this step, the client computes a CRC value for each base derived from the token.

***Definition** 3:* Let b_i be a base derived from a token in the deduplication process. The CRC Computing process is mathematically defined as a function CRC : $B \to C$ where B is the set of bases and C is the set of CRC values. for each base (b_i), the CRC Computing process computes a CRC value denoted as $c = \text{CRC}(b_i)$.

This CRC value serves as a unique identifier for the base, enabling fast and straightforward comparisons. The client maintains a local database of these CRC values, keeping track of the bases it has encountered. As storage capacity becomes limited, the CRC value with the lowest frequency of occurrence is replaced with the newly extracted base to optimize space utilization. Specifically, the CRC value with the lowest frequency of occurrence is identified using the *MinFreq*(.) function in Algorithm 2.

Before processing a new token, the client checks if the corresponding base's CRC value already exists locally. If so, it indicates that the base is a duplicate and has been uploaded previously. In this case, the client transmits only the CRC value and deviation to the cloud, eliminating the need to send the entire base. This process saves bandwidth and reduces network traffic.

On the other hand, if the base's CRC value is not found locally, it implies it is new and has not been uploaded. The client then stores the CRC value locally and transmits both the encrypted base and the corresponding CRC value and deviation to the cloud. This ensures that all data stored on the cloud is correctly identified and tracked for deduplication purposes.

Using CRC values in data deduplication offers several advantages. It simplifies comparisons between data blocks, making it easier to identify duplicates and reduce storage requirements. Additionally, CRC values are lightweight hashing methods, reducing the computational overhead associated with deduplication operations. Furthermore, using CRC values adds an extra layer of confidentiality, as attackers cannot directly access the raw data stored on the client side. This protects the sensitive content from unauthorized access and potential data breaches.

5) ENCRYPTION

The client encrypts the base to enhance data security further before transmitting it to the cloud. This encryption process utilizes a derived key from the KDC and a robust encryption method. The encrypted base, the deviation, and CRC values are then sent to the cloud.

Definition 4: Let b represent a base derived from the data. The Encryption process is formally defined as a function Encrypt : $B \rightarrow E$, where B is the set of bases and E is the set of encrypted data. For each b, the Encryption process computes an encrypted output denoted as $e = \text{Encrypt}(b, k)$. Encryption adds an extra layer of protection, making it more difficult for unauthorized parties to obtain the original base data. This protection extends to both the communication channel and the cloud storage itself. The choice of encryption method is crucial, as it can significantly impact bandwidth consumption. Therefore, it is essential to carefully consider the trade-off between security and bandwidth usage when selecting an encryption method. We also note that deviations do not contain any sensitive information that necessitates encryption. Therefore, they are transmitted in plain text to reduce the computational overhead on the client. This optimization allows the client to focus its resources on processing and encrypting the base, ensuring that the overall data management process remains efficient and secure.

B. DUPLICATE MANAGEMENT ON CLOUD SIDE

When a client uploads data, the cloud-side processing performs the following actions to identify and handle duplicates efficiently:

1) **CRC Value Check:** The cloud first checks its storage for an existing entry with the same CRC value (c_i) received from the client.
2) **Duplicate Encrypted Base Check (if CRC match)**: If a matching CRC value is found, it indicates a potential duplicate. The cloud retrieves the corresponding encrypted base (e_j) associated with that CRC value and compares it with the received encrypted base (e_i) There are two cases:
   - **Identical Encrypted Bases ($e_i = e_j$):** This confirms a true duplicate. The cloud utilizes a pointer-based approach to reference the existing base (e_j) instead of storing a new copy of the encrypted data (e_i). This pointer points to the location where the identical encrypted base is stored, minimizing storage overhead.
   - **Different Encrypted Bases ($e_i \neq e_j$):** This scenario can occur due to the encryption randomness. The encryption process itself introduces randomness, leading to variations in the encrypted outputs (e_i and e_j) even for identical plain texts. Moreover, someone may not honestly use the key and try to produce the wrong encrypted data to interfere the system. In both cases, the cloud prioritizes storing the data uploaded by the current client. In this case, it appends a new entry to its database for the data with the same CRC value (c_i) but a different encrypted base (e_i). This strategy helps to mitigate Poison attacks, which are further discussed in Section VI.
3) **Classical Deduplication Techniques (if no CRC match)**: If the cloud doesn't find a matching CRC value, it implies the data is likely unique. We also note that there is a potential scenario where the cloud might receive a duplicate encrypted base if the client's local storage becomes full and a CRC value is removed. In both cases, DEDUCT can leverage classical deduplication techniques and add a new entry.

V. PERFORMANCE METRICS

This section outlines the performance metrics utilized to evaluate the proposed method.

A. COST MODEL

The cost model considers data storage on both clients and the cloud. To measure the size of a given dataset, we utilize the function Size(.), which calculates the total number of bits required for storage. Additionally, M_{size} represents the storage capacity of the client.

1) CLIENT STORAGE COST

The client's storage cost primarily depends on the size of the CRC function's output value, denoted by C_{size}. This value represents the number of bits required to represent a single CRC value. The client's storage cost is calculated as follows:

$$S_{\text{client}} = |\text{LC}| \times C_{\text{size}}, \tag{3}$$

where LC is the set of local CRC values stored on the client side. We also note $S_{\text{client}} \leq M_{\text{size}}$.

2) CLOUD STORAGE COST

The cloud stores various data components, including received deviations, CRC values, encrypted bases, and pointers for duplicate received bases. To accurately calculate the storage cost for deviations, it is essential to consider each component of op. The Wagner-Fischer algorithm utilizes three operation types, requiring 2 bits for o_t. The second element (o_i) can be represented using $\log_2(|b|)$ bits. For o_v, storage cost depends on the alphabet set Ω, demanding $\log_2(|\Omega|)$ bits per element. Thus, the storage cost for each deviation is determined

as follows:

$$C_{\text{dev}_t \to b} = |\text{dev}_{b \to t}| \times (2 + \log_2(|b|) + \log_2(|\Omega|). \quad (4)$$

The size of encrypted bases, denoted by E_{size}, is directly influenced by the encryption method employed by the client. Additionally, the cloud must store the received CRC values of each encrypted base. We denote the duplicate received bases by dp; each duplicate needs a pointer to the encrypted stored bases. Each pointer, in this case, requires $\log_2(|S_{\text{EB}}|$ bits where S_{EB} is the set of stored encrypted bases. Thus, the cloud's storage cost can be defined as:

$$S_{\text{cloud}} = (|S_{\text{EB}}| \times E_{\text{size}}) + (|\text{dp}| \times \log_2(|S_{\text{EB}}|)) + (\sum_{t \in T} C_{\text{dev}_b \to t}) + |\text{SC}| \times C_{\text{size}}, \quad (5)$$

where SC is the set of stored CRC values in the cloud after cloud-side deduplication, and |dp| is the number of duplicate bases.

B. ENCRYPTION RATIO

To determine the encryption ratio (E_r) of the system, we compute the portion of storage occupied by all transmitted encrypted bases (T_{EB}) to the size of the raw data.

$$E_r = \frac{\text{Size}(\text{T}_{\text{EB}})}{\text{Size(Raw Data)}}. \quad (6)$$

C. COMPRESSION RATIO

The compression ratio (C_r) measures the data deduplication efficiency in the cloud. It represents the proportion of the cloud's storage size to the size of the raw data after deduplication. This can be expressed as follows:

$$C_r = \frac{S_{\text{cloud}}}{\text{Size(Raw Data)}}. \quad (7)$$

D. BANDWIDTH RATIO

To effectively evaluate the bandwidth efficiency of our data deduplication system, we analyze the bandwidth usage and derive a corresponding ratio. This ratio represents the proportion of bandwidth consumption for data deduplication compared to the raw data size.

The bandwidth usage (BW_U) quantifies the total amount of data transmitted over the communication channel. It encompasses the size of deviations, encrypted bases, and CRC values and can be expressed as:

$$\text{BW}_\text{U} = (|T_{\text{EB}}| \times E_{\text{size}}) + (|\text{dp}| \times C_{\text{size}}) + (\sum_{t \in T} C_{\text{dev}_b \to t}), \quad (8)$$

where T_{EB} refers to the set of encrypted bases sent through the channel. It is important to note that the value of $|T_{\text{EB}}|$ might not be equal to the size of the encrypted base set on the cloud side, represented as S_{EB}. This is because the cloud performs deduplication internally, reducing the number of encrypted bases transmitted.

The bandwidth ratio (BW_r) is defined as the proportion of BW_U to the size of the raw data:

$$\text{BW}_\text{r} = \frac{\text{BW}_\text{U}}{\text{Size(Raw Data)}}. \quad (9)$$

A lower value of BW_r indicates more efficient bandwidth utilization for the system.

VI. SECURITY ANALYSIS

In this section, we evaluate the security of DEDUCT, considering two primary adversaries: a malicious CSP and a malicious user. We assume that data transmission is protected through a secure communication protocol, such as SSL/TLS. Further details regarding identification, authentication, and secure communication are beyond the scope of this work and will not be discussed here.

Case 1. Assume that the adversary is the malicious CSP.

The confidentiality of data is fundamentally protected by employing robust encryption algorithms, such as AES, which resist brute-force attacks. Even with a powerful adversary, the data remains secure due to the strength of these encryption mechanisms. However, a potential threat arises from the CSP attempting an offline brute-force attack on CRC values. Using tokenization to split data into variable-length tokens introduces an additional layer of complexity for the CSP. The CSP is unaware of the exact size of the raw data, requiring it to consider all possible combinations for different data sizes to initiate a brute-force attack. Furthermore, the CSP's lack of awareness regarding the manipulated data type adds another layer of protection. The CSP can only attempt brute-force attacks based on the CRC values, which may not provide sufficient information to decrypt the data accurately.

To assess the vulnerability to brute-force attacks based on CRC collisions, we introduce a formula to estimate the potential collisions for a given data size (n bits) and a fixed CRC size (k bits, e.g., CRC-8). It considers the total number of possible unique data values (distinct bit combinations) 2^n and the number of possible unique CRC values 2^k:

$$\text{Col}(n, k) = \frac{2^n - 2^k}{2^k}. \quad (10)$$

For instance, with a data size of 16 bits and CRC-8, the potential number of collisions would be 255.

The occurrences of duplicate CRCs for a specific entry, denoted by Occ(n,k), is calculated as:

$$\text{Occ}(n, k) = \text{Col}(n, k) + 1. \quad (11)$$

The probability of a successful brute-force attack by the CSP, considering occurrences of identical CRCs across different bit lengths, can be expressed as:

$$P_{\text{Success}} = \frac{1}{\sum_{n=1} \text{Occ}(n, k)}. \quad (12)$$

This probability is exceptionally low. For instance, considering a threshold of 24 bits, the CSP would need to construct all combinations from 1 bit to 24 bits and compare the CRCs.

The resulting probability would be $\frac{1}{131078} = 0.0000076290$, indicating a highly unlikely scenario.

Case 2. Assume that the adversary is a malicious user.

As previously outlined, the DEDUCT system model intentionally withholds any explicit response regarding the existence or non-existence of clients' information. Therefore, the malicious user cannot conduct online brute-force attacks on data.

To mitigate the risk of a Poison attack, the system employs a strategy that compares the CRCs of uploaded data. The cloud system handles storage differently if the CRCs match, but the encrypted values differ. Instead of pointing to an existing entry with a different encrypted value, it creates a distinct entry for the new user. This effectively addresses Poison attacks. However, this strategy introduces a potential vulnerability known as a Sybil attack [46]. A malicious actor could create multiple fake identities or entries, exploiting the system's tendency to create new entries for varied encrypted values. This could flood the system with fake entries and pose a risk to its available resources. To prevent users with fake entries from overwhelming the system, one viable strategy is to set a threshold value for each user. This would limit the number of data uploads with the same CRC but different encrypted values a user can perform. Additionally, implementing effective identity management strategies is recommended to prevent users from creating multiple accounts.

VII. PREVENTING INFORMATION LOSS

Data deduplication offers significant storage optimization benefits, but maintaining data integrity throughout the process is a critical challenge. Inaccurate identification of duplicates or inadvertent data alteration can lead to significant information loss. Our proposed method addresses this challenge through a comprehensive approach that prioritizes efficient deduplication and information preservation.

A. ACCURATE DUPLICATE IDENTIFICATION AND CONTEXT PRESERVATION

DEDUCT leverages a multi-step process to ensure accurate duplicate identification while preserving contextual information within textual data. This process involves Tokenization, Transformation, and CRC Computation. By combining these techniques, DEDUCT can accurately identify duplicate data segments, even when subtle variations exist. Additionally, the approach prioritizes retaining the semantic meaning and context of the original data. This is achieved by:

- **Preserving token order:** Maintaining the original order of tokens within each segment ensures contextual relationships are not compromised.
- **Accounting for deviations:** DEDUCT captures any deviations from the ''normalized'' form using transformation techniques. These deviations are stored alongside the CRC value, allowing for the reconstruction of the original data when necessary.

B. EFFICIENT DATA TRANSMISSION AND SECURITY MEASURES

DEDUCT optimizes data transmission to the cloud by focusing on unique data segments. This approach minimizes bandwidth usage and reduces the risk of information loss associated with network issues:

- **Selective Transmission:** Only unique data segments and their corresponding CRC values and deviations are transmitted.
- **Data Encryption:** Sensitive textual data is encrypted before transmission, safeguarding its confidentiality and integrity.

By transmitting only essential data and employing encryption, DEDUCT minimizes the risk of information loss due to:

- **Redundant storage:** Duplicates are not stored unnecessarily, preserving the richness and originality of the data.
- **Network congestion or packet loss:** Reduced data transmission mitigates the potential for data loss during transmission.
- **Data breaches or unauthorized access:** Encryption protects sensitive information from unauthorized access, preventing information loss due to security breaches.

C. ADAPTABILITY AND SCALABILITY

DEDUCT's architecture is designed to handle diverse textual datasets. Its scalability ensures efficient deduplication even as data volumes increase. This adaptability minimizes the risk of information loss associated with methods that may not be well-suited to specific textual data types. Through these strategies, DEDUCT aims to minimize the risk of information loss during the deduplication process, preserving the deduplicated data's quality, integrity, and context.

VIII. PERFORMANCE EVALUATION

This section presents the experimental evaluation of our proposed method using Touchdown [13] dataset. The evaluations are performed on a system with an Intel(R) Core(TM) i7-10510U CPU running at a base frequency of 1.80 GHz, a maximum frequency of 2.30 GHz, and 16GB of RAM. The evaluation program is implemented using Python 3.10.4.

We first analyze the time complexity of each component individually. Subsequently, we determine the impact of threshold values on the defined metrics. This allows us to identify the optimal threshold value and investigate the compression ratio the proposed method achieves with varying configurations. Next, we compare the performance of our method in terms of compression ratio to various deduplication techniques. We also assess the bandwidth consumption ratio and present the encryption ratio obtained by the proposed method.

A. COMPLEXITY ANALYSIS

In this section, we will examine the time complexity of various components in our proposed methods. We note that our complexity analysis focuses on the techniques used

within DEDUCT. The complexity of KDC involves secure key management and distribution, which are independent of the encrypted data and introduce additional complexities beyond the scope of this section.

1) TOKENIZATION

We use the NLTK library for tokenization [36]. This function's time complexity is $O(n)$, where n is the length of the input text. This means that the processing time will increase linearly with the size of the input text, indicating a linear time complexity.

2) LEMMATIZATION

We utilize the WordNetLemmatizer [35] function for lemmatization, which employs a Python dictionary. This dictionary facilitates direct lemma retrieval based on the word, part of speech, and offset. Since dictionary lookups in Python are typically $O(1)$ on average, the time complexity for lemma lookup using the WordNetLemmatizer is also $O(1)$. By leveraging this dictionary-based implementation, the WordNetLemmatizer achieves constant-time performance for lemma lookups, regardless of the size of the WordNet lexicon or the number of words being lemmatized.

3) WAGNER_FISCHER ALGORITHM

The Wagner-Fischer algorithm employs dynamic programming and a matrix to determine the minimum edit distances between substrings. Its time complexity depends on both matrix initialization and computation. The initialization phase involves filling a matrix of size $(m + 1) \times (n + 1)$ with initial values, requiring $O(mn)$ time. The computation step involves accessing each cell to calculate the minimum distance by considering adjacent cells, necessitating $O(m \times n)$ time. Therefore, the algorithm's overall time complexity is $O(m \times n)$, where m and n represent the input strings' lengths.

To improve the algorithm's efficiency, Ukkonen's [34] enhancement offers a time complexity of $O(n + d^2)$, where n is the longer string's length, and d represents the edit distance. In DEDUCT, the value of d is bounded by τ. Thus, the time complexity can be considered $O(n + \tau^2)$. Future work can include optimizations that enhance the algorithm's efficiency in various applications by achieving better time complexity.

4) CRC COMPUTING

The time complexity of computing a CRC is $O(n)$, where n represents the length of the input data. This linear time complexity arises from the iterative process of performing bit-wise operations, such as XOR (exclusive OR) and bit shifts, to calculate the CRC value. During CRC computation, each bit or byte of the input data is processed individually, resulting in several iterations proportional to the input size. Consequently, the time required to compute the CRC increases linearly with the length of the input data.

5) ENCRYPTION

Block ciphers, including DES (Data Encryption Standard) [32], Triple DES [33], and AES (Advanced Encryption Standard) [31], are widely used cryptographic algorithms known for their ability to provide secure data encryption. One notable characteristic of these block ciphers is their time complexity, which can be considered $O(1)$. The time complexity of $O(1)$ implies that the execution time of these block ciphers remains constant regardless of the input size; unlike many other algorithms where the execution time scales linearly with the input size, block ciphers operate on fixed block sizes and perform a fixed number of operations, regardless of the input length. This constant-time execution ensures that the encryption process remains efficient even for large amounts of data.

6) CLOUD-SIDE

When client storage is limited, it can lead to duplicate data being received by the cloud. To prevent this, the cloud efficiently searches for existing data by employing a hash table where each CRC value serves as a key, and its corresponding encrypted data is the associated value. This approach ensures that search, insertion, and deletion operations have an average time complexity of $O(1)$, indicating constant time complexity. Assigning a pointer has a constant time complexity of $O(1)$, regardless of the size or complexity of the assigned data.

When retrieving the original token on the cloud, it is necessary to consider the complexity of the permitted operations in the Wagner-Fischer algorithm that generates the deviations. Deleting and inserting elements have a time complexity of $O(n)$, where n is the length of the string, as they involve shifting subsequent elements. On the other hand, replacing an element has a constant time complexity of $O(1)$ since it directly modifies the element at the specified index. The overall time complexity of reconstructing the token depends on the number of operations in the deviation. The retrieving process involves iterating through each operation in a loop, which takes a time complexity of $O(m)$ where m stands for the number of operations. During this loop, the string elements are combined through concatenation operations, each taking a time complexity of $O(n)$. Considering that concatenation occurs at every iteration, its overall contribution to the total time complexity can be estimated as $O(m \times n)$, where m is the number of operations and n is the maximum length between the two input strings. Considering the complexities of the allowable operations and the concatenation, the overall time complexity for retrieving the original token can be summarized as $O(m \times n)$.

B. THRESHOLD COMPARISON

In this section, we investigate the impact of threshold values on two metrics, the bandwidth ratio (BW_r) and compression ratio (C_r). Our analysis involves two threshold values and assumes the utilization of CRC-8 and CRC-16 methods along with DES-64 and AES-128 encryption methods. Table 2

TABLE 2. The effect of the threshold value (τ) value on compression ratio (C_r) and bandwidth ratio (BW_r) with different CRC values.

			C_r		$\mathrm{BW_r}$	
Client's Storage Ratio	CRC	Encryption	$\tau < 2$	$\tau < 6$	$\tau < 2$	$\tau < 6$
0.0001	CRC-8	AES-128	0.3492	0.3495	0.537	0.5350
0.001	CRC-8	AES-128	0.3492	0.3495	0.246	0.2464
0.01	CRC-8	AES-128	0.3492	0.3495	0.242	0.2427
0.1	CRC-8	AES-128	0.3492	0.3495	0.242	0.2427
0.0001	CRC-16	AES-128	0.3509	0.3511	1.0848	1.0885
0.001	CRC-16	AES-128	0.3509	0.3511	0.4764	0.4760
0.01	CRC-16	AES-128	0.3509	0.3511	0.4494	0.4499
0.1	CRC-16	AES-128	0.3509	0.3511	0.4494	0.4499

summarizes the findings. The client's storage ratio refers to the proportion of data that can be stored locally. For instance, a ratio of 0.1 implies that the client's local storage can accommodate up to 0.1 times the data's size that needs to be uploaded. Table 2 shows how varying storage allocations influence $\mathrm{BW_r}$. As more storage is allocated to the client, the $\mathrm{BW_r}$ declines until it reaches an optimal point. The compression ratio of the proposed system is affected by the chosen threshold value, but the observed difference is relatively minor and can be disregarded.

Selecting a higher threshold value generally results in more deviations and tokens matching the same base. Conversely, a smaller threshold value yields a more effective compression ratio. This efficiency stems from the lower cost of each deviation in smaller tokens. Refining the definition of deviations could potentially alter this trend.

C. BANDWIDTH RATIO

Fig.7 shows the bandwidth ratio ($\mathrm{BW_r}$) for different local storage allocations. The client stores the CRC value of bases locally to save bandwidth usage and only transmits the CRC value to the cloud upon identifying a duplicate base. However, if the client's storage becomes full, it must discard a base with the lowest frequency. This could lead to a duplicate base being classified as new if it was previously transmitted to the cloud but is no longer available in local storage. This directly impacts $\mathrm{BW_U}$, which in turn influences $\mathrm{BW_r}$.

As the client's storage capacity expands for various configurations, we consistently observe a corresponding decline in $\mathrm{BW_r}$. This is because providing adequate storage space allows each encountered CRC value to be stored locally, enabling the client to maintain a comprehensive record of all data and effectively identify duplicates. By allocating local storage that is approximately 0.001 times the size of the Touchdown dataset, it can perform more efficiently and reduce bandwidth usage by almost 67% on average and 78% in the best case scenario.

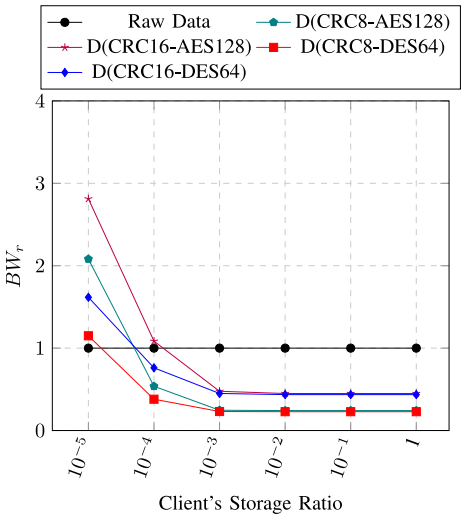


FIGURE 7. The comparison of $\mathrm{BW_r}$ for DEDUCT with different configurations.

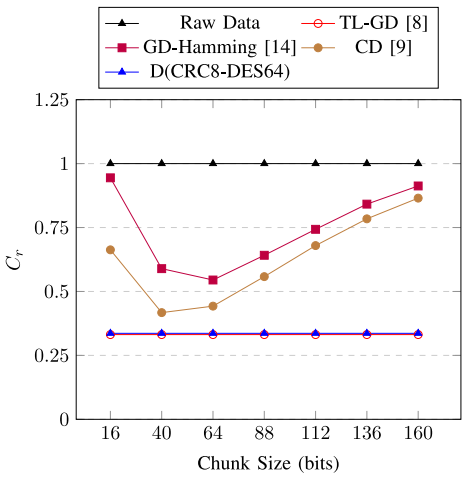


FIGURE 8. The C_r of the proposed method with different configurations.

D. COMPRESSION RATIO

Fig.8 shows the compression ratios obtained with various configurations. By employing CRC-8 and DES-64, the Touchdown dataset can be compressed up to 67%. Even when alternative algorithms such as CRC-16 and AES-128 are utilized, substantial compression can still be achieved, reaching approximately 65%.

Fig.9 compares the compression ratios of the proposed method to TL-GD, GD-Hamming [14], and CD. Comparatively, TLGD and DEDUCT outperform the state-of-the-art. While TLGD exhibits slightly higher compression

ratios, it lacks security measures and bandwidth reduction capabilities. Conversely, DEDUCT achieves remarkable compression ratios while safeguarding data integrity and reducing bandwidth usage. It is worth noting that the threshold value remains constant in each configuration, and the size of the deviations does not change. Additionally, the number of pointers to the encrypted bases depends on the total number of stored encrypted bases.

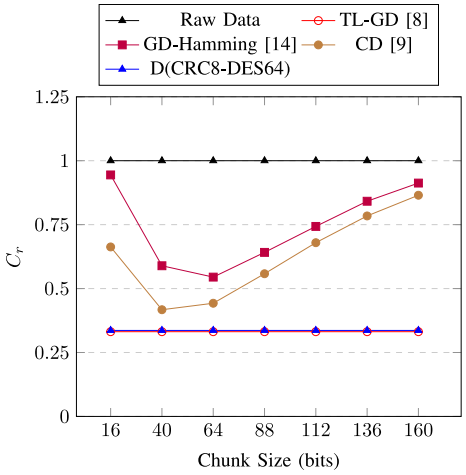


FIGURE 9. The comparison of C_r for different deduplication methods.

E. ENCRYPTION RATIO

The encryption ratio (E_r) of the proposed method is shown in Fig.10. The encryption size and the CRC value influence this ratio. The transmitted encrypted data directly relates to the client's local storage, which is determined by the size of the CRC values. As the storage capacity increases, the number of transmitted encrypted data decreases. Increasing the client's storage capacity is crucial to enhance data transfer efficiency. This reduces the amount of encrypted data transmitted, as only CRC values are sent for duplicate bases. Consequently, a more efficient process is achieved. The figure highlights the relationship between storage capacity and data transmission efficiency.

F. PROCESSING TIME

As illustrated in Fig.11, the DEDUCT system distributes the workload of secure data deduplication between the client and server. The client handles tasks like data splitting, transformation, and encryption, while the server focuses on duplicate data verification using CRC values and storage management with pointers. This breakdown typically leads to a higher client processing time due to encryption and transformations. Factors like data size, duplication ratio, and encryption complexity further influence the workload split. However, increasing client storage can optimize overall performance. With more storage, the client can maintain a larger CRC value cache, enabling it to check for duplicates locally before sending data to the server. This reduces server

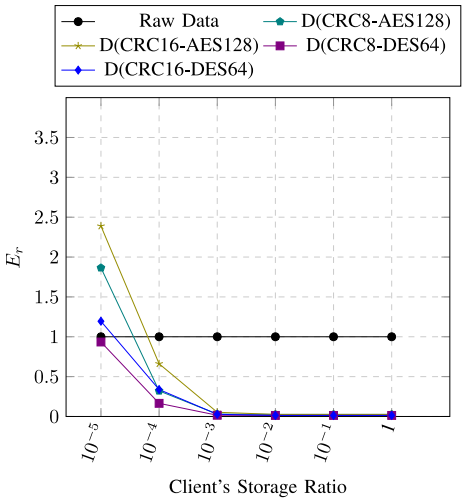


FIGURE 10. The comparison of E_r for the the proposed method.

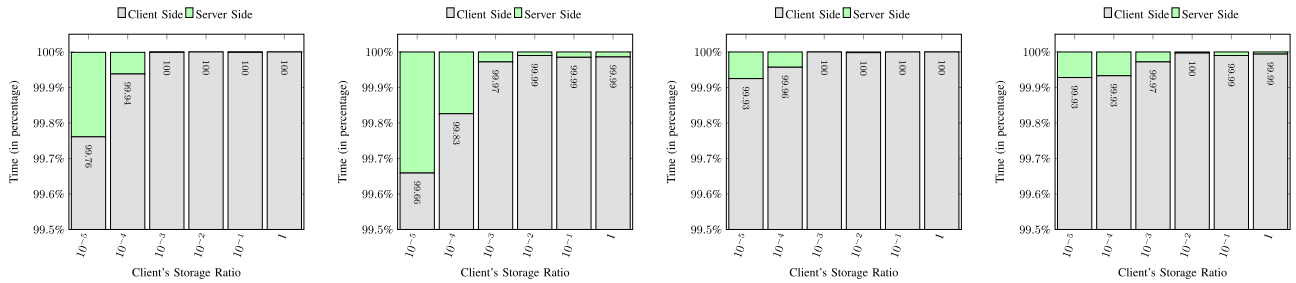
verification workload, as shown in the trend between client storage ratio and server workload in Fig.11. A client-side storage ratio of 100% indicates that server processing time becomes negligible compared to the client-side workload. However, it's important to remember that the server is still responsible for essential tasks like managing storage pointers and performing minimal verifications, even in this scenario. It's also crucial to consider the trade-offs of increased client storage, including the cost of hardware upgrades, potential security risks for sensitive data stored locally, and the processing power limitations of client devices, especially for complex encryption algorithms or large datasets.

IX. RELATED WORKS

This section investigates related works in four categories: Deduplication and privacy, popularity-based encrypted deduplication, mitigating side-channel risks in deduplication systems, and generalized deduplication and privacy concerns.

A. DEDUPLICATION AND PRIVACY

Classic data deduplication methods have raised concerns about data privacy [3]. Traditional encryption algorithms pose challenges for deduplication since they make encrypted data indistinguishable from random bits, making it difficult to identify identical messages. Convergent encryption (CE) [19] was introduced as a solution to achieve encrypted deduplication by deriving encryption keys from the data content. This enables deterministic encryption and ensures that identical messages produce identical ciphertexts. However, CE only offers confidentiality guarantees for unpredictable data, leaving predictable data vulnerable to offline brute-force attacks [6]. Additionally, given sufficient time and resources, the cloud service provider could break the encryption and gain access to the exact information stored in each user's outsourced data [40]. Yang et al. [49] proposed a data deduplication scheme using Boneh-Goh-Nissim cryptosystem and bloom filters. Their approach aims to achieve tag



(a) D(CRC8-AES128) (b) D(CRC16-AES128) (c) D(CRC8-DES64) (d) D(CRC16-DES64)

FIGURE 11. The Portion of total time spent on the client and cloud-side ($\tau < 6$).

consistency, confidentiality, access control, and resistance to brute-force attacks in cloud storage. However, latency remains a concern in its implementation. To further address the challenges of deduplication and privacy, [45] introduces a secure data-sharing scheme that integrates data deduplication and sensitive information hiding. Wildcard substitution is employed in electronic medical records to enhance privacy and deduplication efficiency. Moreover, multiple key servers are utilized to mitigate the risk of brute-force attacks and single-point-of-failure scenarios.

Recent advancements in the data deduplication have focused on optimizing Maximum Likelihood Estimation (MLE) specifically for deduplicating file chunks rather than entire files, enhancing deduplication efficiency [25]. Password-Authenticated Key Exchange (PAKE)-based protocols have also been introduced to facilitate secure key sharing and determination on the client-side [7], [26]. Alternative privacy-enhancing mechanisms, such as Multi-Key Revealing Encryption (MKRE) [39], have been proposed to address the challenges of deduplication while preserving privacy. By using MKRE, the encryption scheme becomes more resistant to attacks attempting to break the encryption. However, the security claims of MKRE have only been proven in the programmable random oracle model, which may not accurately represent real-world scenarios. A secure cloud auditing scheme that supports data deduplication with efficient ownership management was proposed by Wang et al. in [51]. This scheme employs a lazy update strategy to efficiently manage data ownership changes. The cloud maintains a flag determining whether an update is necessary, effectively reducing update frequency and computation overhead.

B. POPULARITY-BASED ENCRYPTED DEDUPLICATION

Data deduplication systems often face a trade-off between data security and storage efficiency. To address this challenge, researchers have explored popularity-based encrypted deduplication schemes. These schemes treat popular data, such as widely shared songs or movies, differently from unpopular data, like medical records or scientific research results. In popularity-based encrypted deduplication schemes, only popular data is encrypted using convergent encryption [19] and subjected to deduplication, while unpopular data is randomly encrypted to ensure semantic security [38]. Existing schemes often rely on a trusted third party to store deterministic tags that record data popularity. However, this reliance on a trusted third party introduces a security vulnerability. If the trusted third party is compromised, the deterministic tags become accessible, enabling offline brute-force attacks to reveal data content [43]. Reference [48] utilizes a double-layer encryption approach for less popular data, allowing Cloud Storage to verify the correctness of inner-layer convergent ciphertext. The outer-layer encryption employs PRP, symmetric encryption, and XOR, leading to reduced computational costs for users and cloud storage.

C. MITIGATING SIDE-CHANNEL RISKS

Harnik et al. were the first to propose a method to safeguard against side-channel attacks by malicious users [37]. They introduced the concept of employing a random threshold for uploads to prevent attackers from inferring the presence or absence of a file on the cloud. The server randomly selects a threshold for each data chunk, and client-side deduplication is only enabled when the number of file uploads surpasses this threshold. This approach prevents an attacker from determining the non-existence of a file. Alternative methods involve randomization of thresholds during operation [10] or their determination based on game-theoretic optimization [11]. Armknecht et al. [12] studied the trade-offs between security and efficiency, proposing a randomized response technique to preserve privacy. Another approach, known as ZEUS [41], necessitates the client simultaneously request the storage of two chunks. The server's response to these requests is deterministic, meaning that if one or both chunks already exist, the user will be prompted to upload a combination of the two. While this indicates the server's possession of at least one chunk, it does not reveal which one specifically. Building upon this concept, RARE [17] further enhances protection by randomly requesting the user to upload either both chunks or a combination of the two whenever at least one chunk is detected on the server. As a result, attackers face more significant challenges in accurately determining whether a particular chunk is stored in the cloud. To further address

client-side deduplication, CIDER [18] extends the principles introduced by RARE to encompass the simultaneous storage of more than two chunks. This method enables users to request the storage of multiple chunks simultaneously. Before enabling client-side deduplication, users must include two fingerprints in each storage request to facilitate proper randomization of responses and ensure that no individual chunk can undergo client-side deduplication.

D. GENERALIZED DEDUPLICATION AND PRIVACY CONCERNS

Sehat et al. introduced Yggdrasil [15], an innovative privacy-preserving deduplication mechanism that tackles the programmable random oracle challenge. Their approach involves clients applying information-theoretic transformations to their data before uploading it to the cloud service provider. Bonsai [44] addresses the limitations of Yggdrasil by presenting a novel approach that eliminates the need for local storage on the client side, reduces the search complexity for the CSP, and enhances privacy safeguards. Unlike Yggdrasil, which assumes that the CSP is unaware of the client's data distribution, Bonsai does not make this assumption, making it more adaptable for handling data with diverse characteristics. HEKATE [16] is introduced as a valuable resource for analyzing the impact of different configurations on throughput and deduplication ratios. The significance of this tool lies in its ability to assist users in evaluating classic or generalized deduplication approaches based on the specific data characteristics of their workloads without requiring a complete system implementation. By employing HEKATE, designers, and administrators gain insights into how a particular configuration will influence system performance and the extent of storage reduction that can be anticipated.

X. CONCLUSION

This paper presents DEDUCT, a textual deduplication technique that leverages generalized deduplication and client-side preprocessing to significantly enhance cloud storage efficiency and data security. DEDUCT demonstrates notable improvements in these key areas compared to existing state-of-the-art methods. DEDUCT achieves a compression ratio of 66% which translates to direct cost savings and improved scalability for cloud storage solutions, offering increased capacity and reduced financial burden. Moreover, DEDUCT's design is well-suited for resource-constrained devices commonly found in the Internet of Things (IoT). This adaptability addresses crucial needs in resource-limited environments where efficient data handling is critical. While the evaluation focused on the Touchdown dataset, DEDUCT's applicability extends to broader domains. Its strengths in efficiently deduplicating large textual datasets make it highly relevant to IoT, mobile, and embedded systems, where storage and bandwidth are often limited. DEDUCT's flexibility and resource-friendly approach offer valuable solutions for these areas.

We aim to enhance client-side preprocessing techniques for future work by utilizing natural language processing and machine learning algorithms. Employing advanced tokenization and lemmatization algorithms can enhance the accuracy of near-duplicate data identification while reducing computational overhead. Additionally, energy efficiency is paramount in resource-constrained environments like IoT and edge computing devices, and DEDUCT can be further optimized to address this concern.

REFERENCES

[1] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel, "Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 3674–3683, doi: 10.1109/CVPR.2018.00387.

[2] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou, "A comprehensive study of the past, present, and future of data deduplication," *Proc. IEEE*, vol. 104, no. 9, pp. 1681–1710, Sep. 2016, doi: 10.1109/JPROC.2016.2571298.

[3] P. Prajapati and P. Shah, "A review on secure data deduplication: Cloud storage security issue," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, no. 7, pp. 3996–4007, Jul. 2022, doi: 10.1016/j.jksuci.2020.10.021.

[4] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *ACM Trans. Storage*, vol. 7, no. 4, pp. 1–20, Jan. 2012, doi: 10.1145/2078861.2078864.

[5] OpenDedup. (2023). *OpenDedUp*. Accessed: Aug. 6, 2023. [Online]. Available: http://opendedup.org./

[6] S. Keelveedhi, M. Bellare, and T. Ristenpart, "DupLESS: Server-Aided encryption for deduplicated storage," in *Proc. 22nd USENIX Secur. Symp. (USENIX Secur.)*, 2013, pp. 179–194.

[7] J. Liu, N. Asokan, and B. Pinkas, "Secure deduplication of encrypted data without additional independent servers," in *Proc. ACM SIGSAC Conf.*, Oct. 2015, pp. 874–885, doi: 10.1145/2810103.2813623.

[8] K. Ghassabi, P. Pahlevani, and D. E. Lucani, "Deduplication of textual data by NLP approaches," in *Proc. IEEE 97th Veh. Technol. Conf. (VTC-Spring)*, Florence, Italy, Jun. 2023, pp. 1–6, doi: 10.1109/vtc2023-spring57618.2023.10199538.

[9] K. Jin and E. L. Miller, "The effectiveness of deduplication on virtual machine disk images," in *Proc. Israeli Exp. Syst. Conf.*, May 2009, pp. 1–12, doi: 10.1145/1534530.1534540.

[10] S. Lee and D. Choi, "Privacy-preserving cross-user source-based data deduplication in cloud storage," in *Proc. Int. Conf. ICT Converg. (ICTC)*, Oct. 2012, pp. 329–330, doi: 10.1109/ICTC.2012.6386851.

[11] B. Wang, W. Lou, and Y. T. Hou, "Modeling the side-channel attacks in data deduplication with game theory," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Sep. 2015, pp. 200–208, doi: 10.1109/CNS.2015.7346829.

[12] F. Armknecht, C. Boyd, G. T. Davies, K. Gjøsteen, and M. Toorani, "Side channels in deduplication," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Apr. 2017, pp. 266–274, doi: 10.1145/3052973.3053019.

[13] H. Chen, A. Suhr, D. Misra, N. Snavely, and Y. Artzi, "TOUCHDOWN: Natural language navigation and spatial reasoning in visual street environments," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 12530–12539, doi: 10.1109/CVPR.2019.01282.

[14] R. Vestergaard, Q. Zhang, and D. E. Lucani, "Generalized deduplication: Bounds, convergence, and asymptotic properties," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2019, pp. 1–6, doi: 10.1109/GLOBECOM38437.2019.9014012.

[15] H. Sehat, E. Pagnin, and D. E. Lucani, "Yggdrasil: Privacy-aware dual deduplication in multi client settings," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2021, pp. 1–6, doi: 10.1109/ICC42927.2021.9500816.

[16] L. Nielsen and D. E. Lucani, "Hekate a tool for gauging data deduplication performance," in *Proc. IEEE 6th Int. Conf. Smart Cloud (SmartCloud)*, Nov. 2021, pp. 67–72, doi: 10.1109/SmartCloud52277.2021.00019.

[17] Z. Pooranian, K.-C. Chen, C.-M. Yu, and M. Conti, "RARE: Defeating side channels based on data-deduplication in cloud storage," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2018, pp. 444–449, doi: 10.1109/INFCOMW.2018.8406888.

[18] R. Vestergaard, Q. Zhang, and D. E. Lucani, "CIDER: A low overhead approach to privacy aware client-side deduplication," in *Proc. GLOBECOM IEEE Global Commun. Conf.*, Dec. 2020, pp. 1–6, doi: 10.1109/GLOBECOM42002.2020.9348272.
[19] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *Proc. Int. Conf. Distrib. Comput. Syst.*, Jun. 2003, pp. 617–624, doi: 10.1109/icdcs.2002.1022312.
[20] R. Vestergaard, D. E. Lucani, and Q. Zhang, "Generalized deduplication: Lossless compression for large amounts of small IoT data," in *Proc. Eur. Wireless 25th Eur. Wireless Conf.*, May 2019, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/8835941
[21] R. Rivest, *The MD5 Message-Digest Algorithm*, document RFC1321, 1992, pp. 179–194.
[22] R. D. Eastlake and P. B. Jones, "US secure hash algorithm 1 (SHA1)," Sep. 2001. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3174, doi: 10.17487/rfc3174.
[23] T. Hansen and D. Eastlake, "US secure hash algorithms (SHA and SHAbased HMAC and HKDF)," May 2011. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6234, doi: 10.17487/rfc6234.
[24] J. S. Sobolewski, *Cyclic Redundancy Check*. Cham, Switzerland: Springer, 2006, pp. 3–33, doi: 10.1007/1-4020-0613-6-4130.
[25] Y. Zhao and S. S. M. Chow, "Updatable block-level message-locked encryption," *IEEE Trans. Depend. Secure Comput.*, vol. 18, no. 4, pp. 1620–1631, Jul. 2021, doi: 10.1109/TDSC.2019.2922403.
[26] J. Liu, L. Duan, Y. Li, and N. Asokan, *Secure Deduplication of Encrypted Data: Refined Model and New Constructions*. Cham, Switzerland: Springer, 2018, pp. 374–393, doi: 10.1007/978-3-319-76953-0-20.
[27] A. Goker and J. Davies, "Web information retrieval," in *Information Retrieval: Searching in the 21st Century*. Cham, Switzerland: Springer, 2009, pp. 85–101.
[28] R. M. Kaplan, "A method for tokenizing text," in *Inquiries Into Words, Constraints and Contexts*. Stanford, CA, USA: CSLI Publications, Jan. 2005, pp. 55–64.
[29] V.I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Sov. Phys. Doklady*, vol. 10, no. 8, pp. 707–710, Jan. 1966. [Online]. Available: https://ci.nii.ac.jp/naid/10020212767
[30] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *J. ACM*, vol. 21, no. 1, pp. 168–173, Jan. 1974, doi: 10.1145/321796.321811.
[31] S. Heron, "Advanced encryption standard (AES)," *Netw. Secur.*, vol. 2009, no. 12, pp. 8–12, Dec. 2009, doi: 10.1016/s1353-4858(10)70006-4.
[32] D. Coppersmith, "The data encryption standard (DES) and its strength against attacks," *IBM J. Res. Develop.*, vol. 38, no. 3, pp. 243–250, May 1994, doi: 10.1147/rd.383.0243.
[33] S. S. Keller, "Modes of operation validation system for the triple data encryption algorithm (TMOVS): Requirements and procedures," Special Publication (NIST SP), Amer. Nat. Standards Inst., Gaithersburg, MD, USA, 1998, doi: 10.6028/nist.sp.800-20.
[34] E. Ukkonen, "Algorithms for approximate string matching," *Inf. Control*, vol. 64, nos. 1–3, pp. 100–118, Jan. 1985, doi: 10.1016/s0019-9958(85)80046-2.
[35] *NLTK: Sample Usage for Wordnet*. Nltk.org. Accessed: Jul. 7, 2023. [Online]. Available: https://www.nltk.org/howto/wordnet.html
[36] *NLTK: NLTK.Tokenize Package*. Nltk.org. Accessed: Jul. 15, 2023. [Online]. Available: https://www.nltk.org/api/nltk.tokenize.html
[37] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Secur. Privacy*, vol. 8, no. 6, pp. 40–47, Nov. 2010, doi: 10.1109/MSP.2010.187.
[38] J. Stanek and L. Kencl, "Enhanced secure thresholded data deduplication scheme for cloud storage," *IEEE Trans. Depend. Secure Comput.*, vol. 15, no. 4, pp. 694–707, Jul. 2018, doi: 10.1109/TDSC.2016.2603501.
[39] S. Zhang, H. Xian, Z. Li, and L. Wang, "SecDedup: Secure encrypted data deduplication with dynamic ownership updating," *IEEE Access*, vol. 8, pp. 186323–186334, 2020, doi: 10.1109/ACCESS.2020.3023387.
[40] K. Akhila, A. Ganesh, and C. Sunitha, "A study on deduplication techniques over encrypted data," *Proc. Comput. Sci.*, vol. 87, pp. 38–43, Jan. 2016, doi: 10.1016/j.procs.2016.05.123.
[41] C.-M. Yu, S. P. Gochhayat, M. Conti, and C.-S. Lu, "Privacy aware data deduplication for side channel in cloud storage," *IEEE Trans. Cloud Comput.*, vol. 8, no. 2, pp. 597–609, Apr. 2020, doi: 10.1109/TCC.2018.2794542.
[42] V. Balakrishnan and L.-Y. Ethel, "Stemming and lemmatization: A comparison of retrieval performances," *Lect. Notes Softw. Eng.*, vol. 2, no. 3, pp. 262–267, Jan. 2014, doi: 10.7763/lnse.2014.v2.134.
[43] P. Puzio, R. Molva, M. Önen, and S. Loureiro, "PerfectDedUp: Secure data deduplication," in *Data Privacy Management, and Security Assurance: 10th International Workshop, DPM 2015, and 4th International Workshop, QASA 2015, Vienna, Austria, September 21–22, 2015. Revised Selected Papers 10*. Springer, 2016, pp. 150–166, doi: 10.1007/978-3-319-29883-2_10.
[44] H. Sehat, A. L. Kloborg, C. Mørup, E. Pagnin, and D. E. Lucani, "Bonsai: A generalized look at dual deduplication," 2022, *arXiv:2202.13925*.
[45] Z. Wang, W. Gao, M. Yang, and R. Hao, "Enabling secure data sharing with data deduplication and sensitive information hiding in cloud-assisted electronic medical systems," *Cluster Comput.*, vol. 26, no. 6, pp. 3839–3854, Dec. 2023.
[46] J. R. Douceur, "The Sybil attack," in *Peer-to-Peer Systems*, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Berlin, Germany: Springer, 2002, pp. 251–260.
[47] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Advances in Cryptology EUROCRYPT 2013*. Berlin, Germany: Springer, 2013, pp. 296–312.
[48] Z. Wang, W. Gao, J. Yu, W. Shen, and R. Hao, "Lightweight secure deduplication based on data popularity," *IEEE Syst. J.*, vol. 17, no. 4, pp. 5531–5542, Dec. 2023, doi: 10.1109/jsyst.2023.3307883.
[49] X. Yang, R. Lu, J. Shao, X. Tang, and A. A. Ghorbani, "Achieving efficient secure deduplication with user-defined access control in cloud," *IEEE Trans. Depend. Secure Comput.*, vol. 19, no. 1, pp. 591–606, Jan. 2022.
[50] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer, "Kerberos authentication and authorization system," Project Athena Tech. Plan Sect. E.2.1, MIT Project Athena, Cambridge, MA, USA, Tech. Rep., Dec. 1987, p. 136. [Online]. Available: http://web.mit.edu/Saltzer/www/publications/athenaplan/e.2.1.pdf and http://web.mit.edu/acs/athena.html
[51] M. Wang, L. Xu, R. Hao, and M. Yang, "Secure auditing and deduplication with efficient ownership management for cloud storage," *J. Syst. Archit.*, vol. 142, Sep. 2023, Art. no. 102953, doi: 10.1016/j.sysarc.2023.102953.
[52] X. Yang, R. Zhang, C. Yue, Y. Liu, B. C. Ooi, Q. Gao, Y. Zhang, and H. Yang, "VeDB: A software and hardware enabled trusted relational database," *Proc. ACM Manage. Data*, vol. 1, no. 2, pp. 1–27, Jun. 2023.
[53] S. S. Patra, S. Jena, J. R. Mohanty, and M. K. Gourisaria, "DedupCloud: An optimized efficient virtual machine deduplication algorithm in cloud computing environment," in *Data Deduplication Approaches*. Amsterdam, The Netherlands: Elsevier, 2021, pp. 281–306.

KIANA GHASSABI received the B.Sc. degree in information technology and the M.Sc. degree in computer science from the Institute for Advanced Studies in Basic Sciences (IASBS), Iran, in 2019 and 2023, respectively. Her research interests include wireless communication, data security, network coding, and the Internet of Things (IoT).

PEYMAN PAHLEVANI received the Ph.D. degree in wireless communication from Aalborg University, Denmark, in 2014. He conducted research with the Department of Computer Science, UCLA, and collaborated with institutions including MIT and Porto universities. He is currently an Assistant Professor with the Institute for Advanced Studies in Basic Sciences (IASBS). His expertise spans wireless communication, network coding, vehicular communications, cooperative networking, and WiFi video streaming. He also contributed to international conferences and reviewed high-impact journals, such as IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY.