Secure and Efficient Deduplication for Cloud Storage with Dynamic Ownership Management

Mira Lee and Minhye Seo *

Department of Cyber Security, Duksum Women’s University, Seoul 01369, Republic of Korea; dsu2007@duksung.ac.kr
* Correspondence: mhseo@duksung.ac.kr

Abstract: Cloud storage services have become indispensable in resolving the constraints of local storage and ensuring data accessibility from anywhere at any time. Data deduplication technology is utilized to decrease storage space and bandwidth requirements. This technology has the potential to save up to 90% of space by eliminating redundant data in cloud storage. The secure data sharing in cloud (SeDaSC) protocol is an efficient data-sharing solution supporting secure deduplication. In the SeDaSC protocol, a cryptographic server (CS) encrypts clients’ data on behalf of clients to reduce their computational overhead, but this essentially requires complete trust in the CS. Moreover, the SeDaSC protocol does not consider data deduplication. To address these issues, we propose a secure deduplication protocol based on the SeDaSC protocol that minimizes the computational cost of clients while leveraging trust in the CS. Our protocol enhances data privacy and ensures computational efficiency for clients. Moreover, it dynamically manages client ownership, satisfying forward and backward secrecy.

Keywords: deduplication; cloud storage; data sharing; message-locked encryption; dynamic ownership update

1. Introduction

In the era of the fourth industrial revolution, digital technology permeates every aspect of our lives, generating vast amounts of client data in real time. The collected data are used in extensive big data analyses, applied in diverse areas including pattern recognition and predictive analytics. Companies leverage client data to formulate effective business strategies, while individuals enjoy personalized services tailored precisely to their needs.

- Social media: Social media platforms generate a wide range of data daily, including posts, photos, and videos. And, they captures client activities and interests for the purpose of delivering personalized content and targeted advertising.
- Search the Internet: Search engines such as Google (90.82%), Yahoo (3.17%), and Bing (2.83%) analyze clients’ search terms and click behavior to improve search results, providing personalized information and advertisements [1].
- Internet of Things: Internet of Things (IoT) devices gather diverse environmental data such as temperature, humidity, and location in smart cities and homes. The collected data are used to improve the client convenience and energy efficiency.

As a large volume of data is rapidly generated and accumulated, discussions have arisen about efficient methods to manage these data, and one of them is cloud services. Cloud services provide a virtual server environment that grants clients access to intangible IT resources, including software and storage, with costs incurred only for actual usage. This not only reduces expenses linked to local equipment management but also mitigates the risk of data loss. Furthermore, cloud services offer the convenience of seamless data access,
enabling clients to manage and retrieve their data from anywhere with an Internet connection. The advent of cloud services has effectively transcended the physical limitations of local computing power and storage.

Deduplication technology is widely employed in cloud storage to minimize service space and lower bandwidth requirements. Deduplication refers to not storing uploaded data twice if they already exist in cloud storage. Instead, the information of the client who uploads the data is linked to the identical data in the cloud storage. Clients who own the same data can access and retrieve them within storage [2]. Data deduplication has the potential to save up to 90% of storage space while providing the same advantages as storing data multiple times [3].

It is important for clients to take into account various security concerns while utilizing the service. Individual clients may express concerns about potential leaks of personal information, while corporate clients may worry about service interruptions or the disclosure of confidential data. To address these security concerns, clients should have the option to directly encrypt and upload their data. However, when clients encrypt and upload data, there is a high likelihood that they might use different secret keys. Encrypting the same data with different secret keys will result in different ciphertexts. When cloud storage attempts to check for data duplicates, it becomes difficult to confirm if different ciphertexts are derived from the same plaintext. Consequently, data deduplication becomes unfeasible. A solution to this problem is the introduction of a new encryption approach for data deduplication, called convergent encryption (CE) [4]. In CE, encryption keys are derived directly from the data themselves. More specifically, the hash value of the data is used as the encryption key, which generates the same encryption key (and thereby the same ciphertext) from the same data. Consequently, it becomes feasible to deduplicate encrypted data for multiple clients sharing the same data.

There are various ways to use cloud storage securely. If the clients uploads the data directly, then the client should encrypt their own data during upload and go through extra steps by themselves. To alleviate this burden, a novel approach was proposed [5], employing a trusted third party called a cryptographic server (CS). In this protocol, the CS receives data from clients and handles the encryption on their behalf, thereby enhancing client convenience. Nonetheless, this protocol comes with several limitations:

- The protocol lacks data deduplication functionality, resulting in an inefficient usage of cloud storage space.
- The protocol requires trust in the CS as it exposes the client’s plaintext to the CS.
- The protocol does not consider specifically updating ownership information concerning stored data in cloud storage.

In the context of cloud storage, data ownership refers to the right of a client to access data stored in the cloud. Clients acquire ownership by uploading data to cloud storage. Furthermore, after storing their data in the cloud, clients can request modifications or deletions. In cases where a client requests data modification or deletion, the cloud storage can remove the client’s information from the group that owns the data. Particularly, the reason for removing the client from the data owner group when a modification request occurs is because cloud storage recognizes the original and modified data as distinct entities. Therefore, to prevent the client from accessing the data before modification, cloud storage needs to delete the client’s information from the group, necessitating a change in the client’s ownership status. Data deduplication requires changing ownership information, which is called a dynamic ownership update. Such changes typically occur in two scenarios: firstly, when clients delete or modify their stored data, leading to a revocation of ownership; and, secondly, when ownership is acquired by uploading data already existing in cloud storage. Dynamic ownership updates in deduplication guarantee that revoked clients cannot access the data and newly added clients are prevented from gaining access to old data [6]. These ownership updates can occur frequently in cloud services, necessitating effective ownership management.
1.1. Contributions

In this paper, we propose a secure and efficient deduplication protocol for cloud storage. Our proposed protocol offers the following advantages:

- **Efficient alleviation of client’s computational costs.** Our study focuses on scenarios where clients upload data directly to cloud storage services, necessitating the encryption of data for secure storage. Our proposed deduplication protocol is based on the secure data sharing in cloud (SeDaSC) protocol [5], which aims to enhance the computational efficiency for clients utilizing cloud services. Similar to the SeDaSC protocol, ours also integrates a third party called a cryptographic server (CS). The CS encrypts data and the CS executes the data deduplication process. And, our proposed protocol demonstrates efficiency in terms of client-side computational cost compared to existing server-side deduplication protocols.

- **Strong assurance of data privacy.** In the SeDaSC protocol [5], as clients transmit plaintext to the CS, there is a requirement for trust in the CS, leading to potential privacy infringements. Our protocol prevents the exposure of data to the CS by having clients blind encrypt the data before transmitting them to the CS. The CS then performs CE on the blind encrypted data, enabling deduplication on the encrypted data in cloud storage. Essentially, our protocol ensures privacy for both the CS and cloud storage.

- **Reduced third-party dependency.** Given that the CS in the SeDaSC protocol has access to data in plaintext, the security of the protocol relies heavily on placing strong trust in the CS. To reduce dependency on the CS, Areed et al. proposed a method where the client employs convergent encryption even when a CS is in place [7]. However, this approach negates the advantage of the CS in reducing the client’s computational overhead. In our protocol, the CS still performs convergent encryption, but the client has the capability to reduce its level of trust in the CS by providing data that are blindly encrypted.

- **Secure data management in cases of dynamic ownership changes.** Existing deduplication protocols using a CS [5,7] do not specifically consider changes in ownership of data (stored in cloud storage) that may occur due to clients modifying or deleting data. Ref. [5] states that, upon revocation of ownership, clients cannot access the data stored in cloud storage. However, the method mentioned assumes that, without proper authentication of being the rightful owner, the client cannot decrypt the data as they possess only encryption key fragments. Hence, the mentioned process differs in dynamically managing ownership to acquire security elements. Clients’ ownership changes are common scenarios in cloud services and data deduplication. Our protocol allows for secure deduplication even in situations where ownership changes occur frequently. By providing dynamic ownership updates, our protocol enhances security, ensuring both forward and backward secrecy.

1.2. Organization

The following sections of the paper are structured as follows. In Section 2, we overview the existing research on secure deduplication protocols. Section 3 describes the background ideas and concepts employed in our proposal. Section 4 discusses the system architecture and security requirements. Section 5 details the construction of our proposed protocol, including its security analysis. Section 6 focuses on the computational analysis of the proposed protocol. Finally, Section 7 concludes the paper.

2. Related Work

The research on secure data deduplication can be divided into server-side deduplication and client-side deduplication depending on the subject that checks and removes data redundancy.
2.1. Server-Side Deduplication

Server-side deduplication is a technology in which cloud storage is the subject of deduplication. When a client uploads data to a server (cloud storage), the server checks whether the data are duplicated. This method is safe for poison attack because the server validates the data collectively before storing them. But, the client always uploads data regardless of whether the data are duplicated, so network traffic increases. Even so, it is difficult for the server to check whether encrypted data are duplicated. If clients with the same data have different encryption keys, different ciphertexts will be generated. To solve this problem, deterministic encryption algorithms have been proposed that use values derived from messages as encryption keys.

In 2002, Douceur et al. introduced convergent encryption (CE), a scheme where the hash value of data is used as an encryption key [4]. In this approach, clients encrypt the result of a cryptographic hash function applied to plaintext using a key, and then upload it to cloud storage. When clients share the same data, they produce an identical hash value. And, if the hash value is used as the encryption key, it can generate the same ciphertext. This characteristic allows for the deduplication of encrypted data. However, since the encryption key is derived from the plaintext, CE is vulnerable to dictionary attack, particularly when the entropy of the plaintext data is low. CE inherently suffers from vulnerabilities to precomputation attacks, where an attacker with encrypted data can make educated guesses about the plaintext data. Bellare et al.’s proposed protocol addresses this issue by utilizing a key server to offer a data deduplication method that is secure against exhaustive brute force attack [8]. The client generates cryptographic keys with the key server through the RSA-OPRF protocol. The client cannot know the key server’s private key, and the key server remains unaware of the client’s CE key.

Before the introduction of CE, the deduplication of encrypted data was not feasible. This was because different clients would generate different ciphertexts for the same data as their encryption keys were different. Starting with the proposed CE [4] in 2002, the MLE [9] was proposed to generate encryption keys from messages. MLE is recognized as the most suitable approach for server-side deduplication. Subsequently, research in server-side deduplication has gained momentum, with a focus on applying it in various environments. In 2013, Puzio et al. proposed a block-level deduplication protocol that solved the client’s key management problem [10]. Block-level deduplication is a method of separating a file into several blocks and encrypting each. In the proposed protocol, the client divides the file into blocks, encrypting each block with CE. In this process, the CE key for the second and subsequent blocks is encrypted with the key of the previous block. Once all the steps are completed, the client stores only the key for the first CE and generates a signature value for each block, which is then uploaded. But, block-level deduplication has the disadvantage of having to remember many random numbers because each file is encrypted with a different key. To address this, the proposed protocol allows clients to remember only the first key, reducing the burden on clients. In 2016, Scanlon proposed a data deduplication approach to reduce digital forensics backlogs [11]. Digital forensic backlogs occur when a significant volume of cases require expert analysis, making it difficult to address each case individually. To solve this problem, he proposed a method of data deduplication and storing data. The proposed method attempted to solve the chronic volume challenge of digital forensics using a centralized data deduplication system. In 2017, Kim et al. proposed the hybrid email deduplication system (HEDS) [12]. This system utilizes single-instance store (SIS) to remove multiple copies at the file level. As a result, the email server stores unique emails and links duplicate emails through pointers. In 2017, Shin et al. proposed a data deduplication protocol based on decentralized server-aided encryption with multiple key servers [13]. Server-aided encryption refers to the help of a server when a client wants to retrieve data. In this context, the server referred to here is a key server. When a client sends a query, the key server encrypts this query before sending it to the cloud. Importantly, the key server does not understand the client’s query, ensuring confidentiality even in a multi-user environment. The proposed protocol does not involve secret key sharing.
among key servers and is a decentralized architecture, making it scalable and suitable for widespread deduplication across various key servers. In 2020, Yuan et al. presented a blockchain-based public auditing protocol that allows for automatic penalties against malicious cloud service providers (CSPs) [14]. The proposed protocol offers compensation to clients when their data integrity is compromised and provides a means to penalize malicious CSPs. To ensure secure and consistent data encryption, the protocol uses MLE [9] with hash and CE with tag check.

In this way, various methods employing server-side deduplication have been proposed to address different scenarios. Some of these studies have focused on scenarios involving a large number of clients or substantial data storage requirements. In 2016, Hur et al. proposed a secure server-side deduplication that remains secure even in environments where the ownership of outsourced data changes frequently [6]. In practical scenarios, when providing cloud storage services, changes in data ownership are likely to occur frequently. For example, a client who previously owned data but had their ownership revoked should no longer have access to data stored in cloud storage. The proposed protocol updates the encryption key each time ownership for data changes, satisfying both forward and backward secrecy. Additionally, the updated encryption keys are selectively distributed to valid owners, and the method strictly manages data access by the owners.

In 2021, Areed et al. proposed a data deduplication protocol for secure data sharing [7] based on a deduplication method for authenticated clients [4]. In this proposed approach, the cryptographic server (CS) generates the CE key on behalf of the client and sends it to the client. The client encrypts the plaintext using the encryption key received from the CS and sends it back to the CS. The CS accesses an access control list (ACL) to confirm data duplication and decide on storage eligibility. This protocol successfully addresses privacy issues [4] but has the limitation of increasing the computational load on the client during this process. Therefore, it appears that there are no existing papers that address the changing environment efficiently and securely, as proposed in this paper. In 2022, Ma et al. introduced a novel server-side deduplication scheme for encrypted data employing hybrid clouds [15]. This approach involves storing data in the public cloud while ownership information and hash code sets for data are stored in the private cloud.

2.2. Client-Side Deduplication

Client-side deduplication is a method in which the client is the subject of deduplication. The client calculates the tag value of the data and transmits it to the server (cloud storage). The server checks whether there is a client’s tag matching in the tag list of the stored data. The server sends the search results to the client. The main feature of the client-side deduplication approach is that clients do not need to encrypt data in the process of verifying that they exist on the server [16]. So, the amount of network transmission is low. However, it is necessary to determine whether the entire data are duplicated by the relatively small tag. Therefore, stored data are vulnerable to poison attack.

In 2011, Halevi et al. proposed a method for proving data ownership using hash tree structures, known as proofs of ownership (PoWs) [17]. In this approach, both the client and cloud construct Merkle trees for their respective data blocks. To build a Merkle tree, data are divided into multiple blocks and arranged as leaf nodes. The leaf nodes are used to create parent nodes until a single root node is generated. Clients can then use this Merkle tree to prove data ownership to the cloud by providing the correct sibling path when requested. But, this approach is sensitive to data size. As data grow, the size of the Merkle tree also increases. And, the cloud must maintain plaintext data to construct the same Merkle tree as the client, which compromises data confidentiality. In 2012, Pietro et al. introduced a secure proof of ownership (s-PoW) that is less dependent on data size [18]. In this method, clients respond to the cloud’s request for a specified number of random bit positions. Nevertheless, the cloud still needs to store plaintext data for the client’s ownership proof challenge. Additionally, the s-PoW exposes plaintext bits during the ownership proof challenge. In 2014, Blasco et al. proposed a PoW using bloom filters
A bloom filter is a probabilistic data structure used to check membership of elements. In this approach, cloud storage uses bloom filters to verify client ownership. However, the bf-PoW is dependent on the size of the data and exposes plaintext bits when issuing ownership proof challenges.

Client-side deduplication aims to reduce network traffic by not sending the entire data. However, during responding to the cloud storage’s challenge to prove ownership, a plaintext issue is exposed. Some proposed solutions aimed to prove ownership using encrypted data. In 2012, Ng et al. proposed a method for proving ownership when ciphertext is stored in cloud storage [2]. The initial uploader stores the plaintext hash value and ciphertext in the cloud storage. Subsequent uploaders verify ownership using the plaintext hash value. However, if the initial client uploads poisoned ciphertext to the cloud storage, subsequent uploaders might lose the original data. In 2013, Xu et al. introduced a secure ownership proof method against exhaustive brute force attack, called hash function client-side deduplication (UH-CSD) [19]. The initial client encrypts plaintext \( m \) with a random key \( K \) to produce \( C_m \), and also encrypts the \( K \) with \( m \) to create \( C_K \). And, the ciphertexts are stored in the cloud storage. Subsequent clients verify ownership, receive \( C_K \) from the cloud storage, and calculate a new \( C_m \), which is then stored in the cloud storage. However, it is vulnerable to poison attack due to the difficulty in proving the relationship between plaintext and ciphertext. In 2015, Manzano et al. proposed a CE-based PoW method (ce-PoW) [19]. Clients split plaintext into chunks and compute CE results. Clients challenge the storage with \( k \) random chunk challenges to prove ownership. However, the client’s calculation and management of CE results for each chunk make the process inefficient. In 2019, Li et al. introduced a client-side encrypted deduplication (CSED) protocol based on MLE [20]. This approach employs a dedicated key server in the MLE key generation process to thwart indiscriminate attacks. Moreover, CSED integrates a bloom-filter-based PoW system to combat illegal content distribution. In 2020, Guo et al. proposed a randomized secure user-to-user deduplication method designed to enhance storage performance in cloud computing services [21]. Clients owning identical copies of data can share the same random value through ElGamal key exchange. In 2021, Al-Amer et al. presented a reinforced PoW protocol [22]. In the proposed method, the cloud storage requests the bit positions of the CE ciphertext. The client must respond with the appropriate blocks for proof of ownership. In 2023, Ha et al. introduced a novel approach called client-side deduplication with encryption key updates [23]. This technique extends the server-assisted encryption method by introducing features like uptable encryption and dynamic proof of ownership. The uptable encryption offers a mechanism to update encrypted data, simplifying the process of modifying existing information or adding new data without the need to decrypt and re-encrypt the entire dataset.

Data deduplication allows for the efficient utilization of storage space, and client-side deduplication helps to conserve network bandwidth. Some approaches store ciphertext in the cloud storage for data confidentiality, while proof of ownership is conducted with plaintext. Nevertheless, these methods still remain vulnerable to poison attacks due to the inability to establish a clear relationship between ciphertext and plaintext. Recently, techniques have emerged that use plaintext transformed into MLE keys for proof of ownership. Client-side deduplication research has mainly focused on ownership proof and encryption-based deduplication. However, research that significantly reduces client-side computational overhead, as our proposed method does, has been lacking.

3. Preliminaries and Background

3.1. Encryption for Secure Deduplication

Clients are required to encrypt and store their data for security purposes. However, if each client encrypts data using their individual secret key, even identical data will produce distinct ciphertexts based on the client. As a result, cloud storage recognizes each ciphertext as a unique object, making data deduplication impossible. To address this issue,
Appl. Sci. 2023, 13, 13270

a proposed encryption method aims to enable clients with identical plaintext to utilize the same secret key.

3.1.1. Convergent Encryption

Douceur et al. proposed a convergent encryption (CE) technique for secure data deduplication in cloud storage [4]. CE is a technique that encrypts data with hash values. With CE, one does not need to share keys in advance because one has the same hash value for the same data. When plaintext data \( m \) is added to the hash function and the output string is \( h(m) \), the encryption key at CE becomes \( h(m) \). Encrypting plaintext \( m \) using encryption key \( h(m) \) creates ciphertext \( C \). Existing data \( m \) can be obtained by decrypting the ciphertext \( C \) using the encryption key \( h(m) \). The hash values of the same data match according to the nature of the hash function, which is a one-way function. In other words, even if the owner is different, if the data are the same, the same hash value can be generated, and if the same encryption key is used, the same ciphertext can be generated. Cloud storage clients can generate ciphertexts using hash values derived from data without the need for key sharing.

- **KeyGen** \((h, m)\). Given a cryptographic hash function \( h \) and plaintext \( m \) as input, a convergent key \( K = h(m) \) is output.
- **Encrypt** \((K, m)\). Given convergent key \( K \) and plaintext \( m \) as input, it produces encrypted data \( C \).
- **Decrypt** \((K, C)\). Given convergent key \( K \) and ciphertext \( C \) as input, it produces decrypted data \( m \).

3.1.2. Message-Locked Encryption

Bellare et al. introduced message-locked encryption (MLE) as a technology for ensuring data integrity in the context of secure data deduplication [9]. MLE is a generic term for encryption techniques that generate encryption keys based on the data themselves. Prior to the introduction of MLE, various encryption methods, including those based on CE, were proposed for data deduplication. However, since the publication of their paper, there has been active research in the field of secure data deduplication, with a particular focus on using cryptographic hash functions to verify data integrity. Currently, there are prominent encryption methods used for secure data deduplication under the MLE umbrella, including CE, hash and CE without tag check (HCE1), hash and CE with tag check (HCE2), and randomized convergent encryption (RCE). In this paper, the MLE technique employed is HCE2.

HCE2 is a technique that builds upon CE and uses keys derived from the data to encrypt and verify integrity through tag consistency. When plaintext data \( m \) is hashed, resulting in the string \( h(m) \), HCE2 uses \( h(m) \) as the encryption key. Using \( h(m) \) to encrypt \( m \) produces ciphertext \( C \), and decrypting \( C \) allows for the recovery of \( m \). In contrast to CE, MLE employs \( C \) as the input to a hash function to generate \( T \), which represents the tag, \( T \) serves to ensure the integrity of the ciphertext. In summary, HCE2, as a representative MLE encryption method, is used to achieve data deduplication with integrity verification by deriving encryption keys from data and utilizing tag consistency. This approach is a crucial component of the broader field of secure data deduplication.

- **KeyGen** \((h, m)\). Given a cryptographic hash function \( h \) and plaintext \( m \) as input, an encryption key for MLE \( K = h(m) \) is output.
- **Encrypt** \((K, m)\). Given an encryption key \( K \) and plaintext \( m \) as input, it produces encrypted data \( C \).
- **Decrypt** \((K, C)\). Given an encryption key \( K \) and ciphertext \( C \) as input, it produces decrypted data \( m \).
- **TagGen** \((h, C)\). Given a cryptographic hash function \( H \) and ciphertext \( C \) as input, it produces an integrity verification tag \( T \), which corresponds to ciphertext \( C \).
3.2. Proofs of Ownership

Halevi et al. proposed a protocol for legitimate data ownership proof. The proposed protocol is a Merkle-tree-based ownership proof protocol [17]. It allows individuals to assert ownership of data that they actually possess by presenting a portion of them to cloud storage. Proof of ownership (PoW) is a protocol used to verify whether a client has legitimate rights to access data stored in the cloud storage. When a client requests to download data, the cloud storage checks the ownership of the client. To establish the legitimacy of ownership, the client must respond appropriately to a challenge presented by the cloud storage. This response enables the cloud storage to determine whether the client has the necessary access rights to the data.

The client uses the following algorithm to encrypt the data and generate the tag value for PoW.

- **KeyGen**($h, m$). Given a cryptographic hash function $h$ and plaintext $m$ as input, an encryption key $K = h(m)$ is produced.
- **Encrypt**($K, m$). Given an encryption key $K$ and plaintext $m$ as input, it produces encrypted data $C$.
- **TagGen**($h, C, b$). Given a cryptographic hash function $h$, ciphertext $C$, and the Merkle tree leaf size parameters $b$ as input, it produces a Merkle tree $MK$ and integrity verification tag $T = MT_b(C)$.

The client stores the previously generated values $K$, $T$, and $MT$. It then sends $T$ and the number of lowest-level leaf nodes to the cloud storage server. If the server finds a matching tag for $T$, it will request ownership proof from the client. In response, the client should provide the requested node and sibling path. However, if no matching tag exists, the cloud storage will request the client to upload encrypted data.

- **Decrypt**($K, C$). Given an encryption key $K$ and ciphertext $C$ as input, it produces decrypted data $m$.

The client transmits $T$ and the number of lowest-level leaf nodes to the cloud storage server. If the cloud storage server finds a matching tag, it will request ownership proof from the client. In response, the client must provide the appropriate node and sibling path as requested by the cloud storage. Upon successful ownership proof, the cloud storage will send encrypted data to the client. The client can then decrypt the data using the decryption algorithm mentioned above.

3.3. Secure Data Sharing in Cloud (SeDaSC) Protocol

Ali et al. introduced the SeDaSC protocol, which aims to reduce client computational overhead when sharing data among authenticated clients in a cloud [5]. In the SeDaSC protocol, the client uploads plaintext data, and a cryptographic server (CS) is responsible for several key operations, while the client does not engage in encryption operations directly. However, it is crucial to fully trust the CS since it has access to plaintext data. The detailed process is outlined below:

1. **Upload.** The client uploads plaintext data. The CS generates an encryption key for the uploaded data. Using the encryption key, the CS encrypts the plaintext and stores the data’s information and client’s information in an access control list (ACL). The CS then splits the generated encryption key into two parts, securely storing one part and transmitting the other part to the client. To further enhance security, the CS overwrites and deletes the initial encryption key. The encrypted data are finally stored in the cloud. The purpose of storing client information in the ACL is to verify the legitimate ownership of data when a download request is made. Splitting the encryption key into two parts prevents any single entity from decrypting the data independently. If it is an initial upload, a key generation process is performed.
   - **KeyGen**(1^L, $h$). Given security parameters $1^L$ and a 256-bit cryptographic hash function $h$ as input, a symmetric key $K = h(\{0, 1\}^{256})$ is produced.
• Encrypt \((m, SKA, K)\). Taking plaintext \(m\), symmetric key algorithms \(SKA\), and symmetric key \(K\) as input, it produces encrypted data \(C = SKA(K, m)\).

• KeyGen for Client \(i\) \((K)\). Given symmetric key \(K\) as input, it generates the key of CS, \(K_i = \{0, 1\}^{256}\), and the key of client \(i\), \(K'_i = K \oplus K_i\).

2. **Download.** The client requests decryption of data stored in the cloud, sending the encrypted data to the CS. The CS uses the information stored in the ACL, along with the symmetric key provided by the client, to recover the encryption key. Since each client has a different \((K_i, K'_i)\) pair, the impersonation of other clients is prevented. If the client sends the correct symmetric key to the CS, it can receive the decrypted data. Alternatively, the client can request the CS to perform both download and decryption. In this case, the client sends the group ID and symmetric key \(K'_i\) to the CS, which retrieves and decrypts the data from the cloud before transmitting them to the client.

• Decrypt \((C, K_i, K'_i, ACL, SKA)\). Given encrypted data \(C\), CS’s key \(K_i\), client \(i\)’s key \(K'_i\), access control list \(ACL\), and symmetric key algorithms \(SKA\) as input, it recovers the encryption key \(K = K_i \oplus K'_i\) and decrypts the data to produce plaintext \(m = SKA(C, K)\).

Areeed et al. used CE, which uses the hash value of plaintext data as an encryption key, to prevent the CS from accessing plaintext [7]. In CE, the CS generates a file key based on the hash value received from the client and the number of clients to share data and divides the file key into half with the client. The client encrypts the data and sends the received half key to the cloud. The proposed protocol prevents the CS from accessing plaintext through CE. In addition, it proposed a communication method that can be used universally compared to existing protocols that consider only authenticated clients. However, the amount of computation increases as the client performs encryption directly. When the CS generates a file key, it does not check the relationship between plaintext data and hash values. Therefore, the cloud cannot confirm the relationship between plaintext data and encrypted data. Because the cloud does not check the integrity of the data, it is vulnerable to poison attacks that store certified plaintext and other data, such as server-side deduplication technology. Therefore, it is difficult to say that the limitations of SeDaSC have been completely solved.

4. **System Model**

4.1. **Entity**

The system model of our proposed secure deduplication protocol is described in Figure 1. The entities in the system model are as follows:

• Client: A client is a person who has ownership by uploading data to cloud storage. Since the data were deduplicated and stored, only the initial client’s data are stored in the storage. The client refers to both the initial and subsequent uploaders that have ownership.

• Cryptographic server (CS): The CS acts as an intermediary between the client and cloud storage. The CS configures the access control list (ACL) with the hash value received from the client. The ACL manages data information stored in the cloud storage and client information that owns it. The CS controls the client’s data access rights based on the ACL. If data need to be stored in cloud storage, CS encrypts the data and sends them to cloud storage.

• Cloud storage: Cloud storage stores data from clients. Cloud storage generates a group key for the data in which the storage request is made to manage the dynamic ownership update. The key is generated independently of a key shared in the previous owner group. The data in which the storage request occurs are re-encrypted with the generated key and stored in the storage. It is assumed that cloud storage is unreliable.
4.2. Security Requirements

The provided points outline essential security and privacy requirements for the proposed protocol. These requirements collectively emphasize the importance of maintaining data privacy and integrity and ensuring secure ownership transitions in the proposed protocol. The detailed requirements are as follows:

- **Data privacy**: Data privacy means that the actual content of the data should be protected from unauthorized access, ensuring that sensitive information within the data remains confidential. The original data remain inaccessible to cloud storage, the CS, and unauthorized clients.

- **Data integrity**: Data integrity involves ensuring that the data stored in the system remain unaltered and reliable. Both the cloud storage and the CS must have mechanisms in place to verify the purity and correctness of the data before storing them or transferring ownership.

- **Forward security**: Forward security is a concept where clients whose ownership has expired must be prevented from accessing data stored on the cloud storage. This ensures that, even after losing ownership rights, clients cannot access data that they previously owned. It aims to prevent unauthorized access, protect the integrity of data, maintain a clear separation of ownership, and ensure that clients cannot access data outside their current ownership scope.

- **Backward security**: Backward security is a concept where clients who have uploaded data to the cloud storage should not be able to access data that were stored before they gained ownership. In other words, even after acquiring ownership rights to certain data, clients should not have access to the historical data records from previous owners.

5. The Proposed Secure Deduplication Protocol

In this section, we propose a secure deduplication protocol based on the secure data sharing in cloud (SeDaSC) protocol [5]. The proposed protocol has the following key characteristics. First, our protocol ensures both computational efficiency for the client and data privacy. By building upon SeDaSC, we alleviate the client from complex operations, entrusting these tasks to a cryptographic server (CS). In the initial SeDaSC protocol, there was a challenge in terms of exposing plaintext to the CS. However, our proposed protocol addresses this concern by introducing message-locked encryption (MLE), making it possible to deduplicate encrypted data. Within our proposal, we employ hash and convergent encryption with tag check (HCE2) within the context of MLE. Second, our protocol includes a feature for dynamic ownership updates. When owners upload or revoke their data, the cloud storage re-encrypts the data using a group key, which is shared by clients who own the same data. This group key is generated by the cloud storage when the owner group changes, and then it is distributed by both the CS and the cloud storage. Table 1 provides a description of the notation used in this paper.
5.1. Initial Data Upload

The client can gain ownership of data in the cloud storage by successfully uploading them. There are two types of data uploads: initial uploads, which involve data not yet stored in the cloud, and subsequent uploads, which pertain to data that are already present. The detailed process of the initial upload is as follows:

Step 1. Upload pre-work. Client $i$ must blind encrypt and send data to CS.

- The client $i$ calculates the hash value $h(m)$ of the message $m$.
- The client calculates $m \oplus h(m) = M$ and a hash value $h(M)$. $M$ is a blind encrypted value sent to the CS. Not only is it generated faster than the encryption operation, but also the characteristics of the plaintext are not revealed to the CS.
- The client randomly selects $r_i$. This will be used to prove the client themselves.
- The client stores $h(M)$ and $r_i$. The $h(m)$ is used to recover $m$ from $M$ when data downloading. The $h(M)$ is used to identify the desired data when requesting an ownership update or data download. The $r_i$ is used to identify the client.
- The client sends an upload request, $M$, and $r_i$ to the CS.

Step 2. Deduplicate data. The CS determines whether the received data are duplicated and processes the data according to the case.

- The CS calculates the hash value $h(M)$ of the received $M$.
- The CS checks whether $h(M)$ and $r_i$ exist in the ACL. The initial upload means no data in the cloud storage. In this case, no information exists in the ACL.
- The CS stores $h(M)$ and $r_i$ in ACL. Also, since there are no data in the cloud storage, CS must encrypt $M$ and send it to the cloud storage.
- The CS encrypts $M$ with the hash value $h(M)$. Encrypting data with a hash value is called MLE. Hash values are always the same for the same data. Thus, MLE can generate the same ciphertext for the same data.
- The CS sends a store request, $h(M)$, and $C = E(M)$ to the cloud storage.

Step 3. Re-encrypt data. Cloud storage generates a group key and re-encrypts the data.

- The cloud storage generates a group key, denoted as $GK_I$, by encrypting the results of XOR operations on $h(M)$ and session $I$ with the cloud storage’s $SK_C$. 

---

Table 1. Notations.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>The plaintext</td>
</tr>
<tr>
<td>$h(\cdot)$</td>
<td>The cryptographic hash function</td>
</tr>
<tr>
<td>$M$</td>
<td>The blind-encrypted data</td>
</tr>
<tr>
<td>$C$</td>
<td>The ciphertext</td>
</tr>
<tr>
<td>$r_i$</td>
<td>The random value of client $i$</td>
</tr>
<tr>
<td>$ACL$</td>
<td>The access control list of CS</td>
</tr>
<tr>
<td>$CTL$</td>
<td>The ciphertext list of cloud storage</td>
</tr>
<tr>
<td>$E(\cdot)$</td>
<td>The symmetric encrypt function</td>
</tr>
<tr>
<td>$SK_C$</td>
<td>The secret key of cloud storage</td>
</tr>
<tr>
<td>$i, j, n$</td>
<td>The client identification name</td>
</tr>
<tr>
<td>session $I, J, N$</td>
<td>The session identification in progress</td>
</tr>
<tr>
<td>$GK_N$</td>
<td>The group key for session $N$</td>
</tr>
<tr>
<td>$RC_N$</td>
<td>The re-encrypted data for session $N$</td>
</tr>
<tr>
<td>$RGK$</td>
<td>The distribution group key</td>
</tr>
<tr>
<td>$RGK_{adder}, RGK_{other}$</td>
<td>The distribution group key for client $n$</td>
</tr>
<tr>
<td>$CRGK_n$</td>
<td>The distribution group key for client $n$</td>
</tr>
</tbody>
</table>
• The cloud storage performs an XOR operation on the ciphertext \( C \) and the group key \( GK_I \). The \( C \) is received from the CS. The result of the XOR operation is a re-encrypted ciphertext \( RC_I \) for session \( I \).

Whenever an ownership update occurs, the cloud storage refreshes a group key \( GK_N \) and a re-encrypted ciphertext \( RC_N \) for the session \( N \).

• The cloud storage stores \( h(M), GK_I, \) and \( RC_I \) in a ciphertext list of cloud storage (CTL).

• The cloud storage generates an RGK to distribute the refreshed group key. Since there are no owners in the previous session, the RGK generates only \( GK_I \).

• The cloud storage sends the generated RGK to the CS and requests it to be sent to the data owner.

**Step 4. Send refreshed group key.** The CS sends the group key to the legitimate client. The CS must send the group key to the client based on \( h(M) \) and \( r_i \) stored in the ACL. And, the client keeps the group key.

• The CS generates the CRGK by XOR operation on the client’s random value \( r_i \) in ACL and the RGK received from the cloud storage. The \( r_i \) is the random value of the client \( i \) stored as owning \( h(M) \) in the ACL.

The CRGK should be generated through the RGK and random value of the client that has ownership of \( h(M) \) among the clients stored in the ACL. If an owner is added, the CRGK should be sent to all clients, and if the owner’s ownership is deleted, the CRGK should be passed to the remaining clients in the group. The clients can prove their ownership by recovering \( GK_N \) from the CRGK using \( r_n \). The client recovers and stores \( GK_I \) from the CRGK received from the CS. The client may use \( GK_I \) when downloading data in the future.

5.2. Subsequent Data Upload

In the case of the initial data upload, the data are not yet stored in the cloud storage. This process involves generating the CS’s ACL and the cloud storage’s CTL. In contrast, subsequent data uploads involve data already present in the cloud storage. In this scenario, client information needs to be added to both the CS’s ACL and the cloud storage’s CTL. Additionally, the cloud storage conducts dynamic ownership updates, which entail refreshing the group key and re-encrypting data when ownership update occurs.

The detailed process of subsequent uploads is as follows:

**Step 1. Upload pre-work.**

• The client \( j \) calculates the hash value \( h(m) \) of the message \( m \).

• The client calculates \( m \oplus h(m) = M \) and a hash value \( h(M) \) to be sent to the CS.

• The client randomly selects \( r_j \) to be used to prove oneself.

• The client \( j \) stores \( h(m), h(M), \) and \( r_j \).

• The client \( j \) sends an upload request, \( M, \) and \( r_j \) to the CS.

**Step 2. Deduplicate data.**

• The CS calculates the hash value \( h(M) \) of the received \( M \).

• The CS checks whether \( h(M) \) and \( r_j \) exist in the ACL. For subsequent uploads, they are divided into two cases.

1. The first case is that client \( j \) uploaded the same data before, but the client does not remember it and re-uploads the data. In this case, \( h(M) \) and \( r_j \) will exist in the ACL of the CS. If so, the CS notifies the client \( j \) that the data are already saved.

2. The second case is that information about \( h(M) \) exists in the CS’s ACL but the client \( j \) is not registered as the owner. In this case, an update of
the ownership group shall be made. The CS stores the client $j$’s random value $r_j$ in the ACL. And, the CS send a group key update request to the cloud storage.

The process of adding ownership in subsequent uploads is described in Section 5.4.

5.3. Data Download

The clients can download data stored in cloud storage at their convenience, whenever they wish. The detailed process of data download is as follows.

Step 1. Request download. The client sends a download request to the CS.

- Client $i$ sends a download request with $h(M)'$, $r_i$, and $GK_N'$ to the CS to download the data.

$h(M)'$ indicates that the data client wants to download. $r_i$ serves as proof of the client $i$’s identity, and $GK_N'$ signifies the client’s involvement in session $N$. The use of a small quotation mark (’) on the values sent by the client visually indicates whether they match the values stored in the CS and the cloud storage.

Step 2. Check ownership. The client sends a download request to the CS.

- The CS checks whether $h(M)'$ and $r_i$ are stored in the ACL. If both $h(M)'$ and $r_i$ exist, the CS will normally perform the download process. However, without $h(M)'$ or $r_i$, CS will send an error message to the client. This is because the client cannot prove ownership to the CS, or the data do not exist in the cloud storage.

- The CS sends a download request, $h(M)'$, and $GK_N'$ to the cloud storage. $h(M)'$ is for identifying data stored in the CTL of the cloud storage, and $GK_N'$ is a group key for session $N$ used to decrypt the re-encrypted data.

Step 3. Cloud storage’s decryption. The cloud storage decrypts the re-encrypted data and sends them to the CS.

- The cloud storage checks whether $h(M)'$ is stored in the CTL. If $h(M)'$ exists, the cloud storage calculates ciphertext $C'$ by performing an XOR operation on the group key $GK_N'$ and re-encrypted data $RC_N$. The $GK_N'$ is received from the CS, and the $RC_N$ is stored in the CTL of the cloud storage.

- The cloud storage sends $C'$ to the CS.

Step 4. CS’s decryption. The CS decrypts the ciphertext and sends to the client.

- The CS decrypts the ciphertext $C'$ as $h(M)'$ to obtain $M'$, and the CS computes the hash value $h(M')$ of the message $M'$.

- The CS checks whether $h(M)$ and calculated $h(M')$ are the same. The $h(M)$ is a value stored in the CS’s ACL. If the two values are the same, it means that they have been decoded normally and will be transmitted to the client. In other cases, it means that an error occurred during the decoding process, and the client will be notified of this error.

- The CS sends $M'$ to the client $i$.

Step 5. Client’s decryption. The client recovers plaintext $m$ from $M$.

- The client recovers the plaintext $m'$ by performing an XOR operation on $h(m)$ and $M'$. The hash value $h(m)$ is stored in the client, and the $M'$ is received from the CS.

If there is no error message received from the CS, $m$ and $m'$ will be the same data.
5.4. Ownership Update

The ownership updates occur in subsequent uploads, i.e., when another client attempts to upload the same data while data are already stored in the cloud storage. During an ownership update, the group key is refreshed and distributed to the clients. Suppose that the session prior to the data upload of the client $j$ is session $I$. When a client $j$ successfully uploads data, it initiates a new session, which is now referred to as session $J$.

**Step 1. Upload pre-work.** The same as Step 1 of the subsequent upload in Section 5.2.

**Step 2. Deduplicate data.**
- The CS calculates the hash value $h(M)$ of the received $M$.
- The CS checks whether $h(M)$ and $r_j$ exist in the ACL.

If information about $h(M)$ exists in the ACL of the CS, but client $j$ is not registered as the owning client, an ownership update should be made.
- The CS stores the random value $r_j$ of client $j$ in the ACL.
- The CS sends a data re-encrypting request with $h(M)$ to the cloud storage. The hash value, denoted as $h(M)$, plays a crucial role in identifying the specific data requested for updating in the cloud storage.

**Step 3. Re-encrypt data.**
- The cloud storage generates a group key, denoted as $GK_J$, by encrypting the results of XOR operations on $h(M)$ and session $J$ with the cloud storage’s $SK_C$. In this context, the encryption with the cloud storage’s secret key is achieved through symmetric key encryption. The hash value $h(M)$ is received from the CS.
- The cloud storage generates re-encrypted data, denoted as $RC_J$, by conducting an XOR operation on $RC_I$, $GK_I$, and $GK_J$. Accordingly, the re-encrypted data $RC_J$ take the form of $C \oplus GK_J = RC_J$.

The previous group key, denoted as $GK_N$, serves to decrypt the ciphertext $C$ within $RC_N$. Essentially, the cloud storage stores only $RC_N$ for efficient storage space utilization. Whenever an ownership update occurs, the cloud storage recovers the original ciphertext $C$ using the group key $GK_N$ from $RC_N$ stored in the CTL. In addition, the cloud creates re-encryption data, $RC_{N+1}$, through an XOR operation on ciphertext $C$ and $GK_{N+1}$. Importantly, a new group key is generated independently of any prior session. This independence arises because when the cloud storage generates a session value, it remains disconnected from previous sessions, ensuring that each session operates independently.
- The cloud storage stores $GK_J$ and $RC_J$ created in the above two processes.
- The cloud storage generates $RGK$ to distribute the refreshed group key to clients.
  - $GK_J = RGK_{adder}$;
  - $GK_J \oplus GK_J = RGK_{other}$.

In subsequent uploads, both the existing data owners from the previous session and new owners are involved. Thus, the cloud storage creates two type of refreshed group keys: the $RGK_{adder}$ for newly added owners and $RGK_{other}$ for existing owners. For newly added owners, $RGK_{adder}$ includes only $GK_{N+1}$, because the new owners are not aware of the previous session $N$. For existing owners, $RGK_{other}$ consists of $GK_N$ and $GK_{N+1}$. And, the prior group key $GK_N$ of $RGK_{other}$ serves to verify the ownership of previous owners.
- The cloud storage sends the generated $RGK_{adder}$ and $RGK_{other}$ to the CS to requests it to be sent to the data owner.
Step 4. Send refreshed group key.

- The CS sends the refreshed group key to the client based on the \( h(M) \) and \( r_i \) stored in the ACL. Suppose that the clients with \( h(M) \) are \( j \) (additional uploader) and \( i \) (existing owner).
  - For additional uploader \( j \):
    1. The CS generates \( CRGK_j \) by performing an XOR operation on the random value \( r_j \) of client \( j \) and the RGK\_adder.
    2. The CS sends \( CRGK_j \) to client \( j \).
    3. The client \( j \) recovers \( GK_j \) from the value received.
       \[ CRGK_j \oplus r_j = GK_j. \]
  - For existing owner \( i \):
    1. The CS generates \( CRGK_i \) by performing an XOR operation on the random value \( r_i \) of client \( i \) and the RGK\_other.
    2. The CS sends \( CRGK_i \) to client \( i \).
    3. The client \( i \) recovers \( GK_j \) from the value received.
       \[ CRGK_i \oplus GK_i \oplus r_i = GK_j. \]

- Both clients store the group key \( GK_j \) for session \( J \).

5.5. Ownership Delete

Clients belonging to the owner group of the data can access the source data. If clients desire to delete the data and revoke ownership, they can initiate this process by sending a request to the CS at any time. In cases where there is a change in group information, it becomes imperative to update the group key and re-encrypted data. This is performed to prevent clients who have previously deleted their ownership from retaining access to the data, ensuring the security and integrity of data management. The process of deleting ownership is as follows:

Step 1. Request ownership revocation. A client sends an ownership release request to the CS.

- The client \( i \) submits an ownership revocation request to the CS, which includes \( h(M) ', r_i \), and \( GK_N ' \). The hash value \( h(M) ' \) specifies which data are owned. The random value \( r_i \) informs the client of who it is. The group key \( GK_N ' \) means that the client belongs to session \( N \) in progress. The small quotation mark \( (') \) on the values sent by the client visually indicates whether they match the values stored in the CS and the cloud storage.

Step 2. Check ownership. The CS checks the client’s ownership,

- The CS checks whether \( h(M) ' \) and \( r_i \) are stored in the ACL.
  If both \( h(M) ' \) and \( r_i \) are provided, the CS proceeds with the ownership deletion process. However, if either \( h(M) ' \) or \( r_i \) is missing, the CS sends an error message to the client. In cases where the client is the last owner of \( h(M) ' \) stored in ACL, the CS additionally sends a group-key delete request to the cloud storage, ensuring proper data management and security. If the client is the last owner of the \( h(M) ' \) stored in the ACL, the CS will also send a group key delete request to the cloud storage.

- The CS sends a download request to the cloud storage, which includes \( h(M) ', GK_N ' \). The hash value \( h(M) ' \) is crucial for identifying information stored in the CTL of the cloud storage. The group key \( GK_N ' \) serves the purpose of decrypting the re-encrypted data in the cloud storage.
Step 3. **Check group key.** The cloud storage checks the client’s group key.

- The cloud storage performs a check to determine whether the stored group key $G_K_N$ matches the $G_K'_N$ received from the CS. If these values match, the cloud storage sends a group key authentication success message to the CS. Otherwise, the cloud storage sends an error message to the CS.

Step 4. **Revoke ownership.** The CS revokes the client’s ownership in ACL.

- If the message received from the cloud storage is successful, the CS removes the $r_i$ from the ACL, then forwards the results to the client. Conversely, in case of a failure message, an error message is sent to the client.

Step 5. **Re-encrypt data.** The cloud storage recreates re-encrypted data. After the completion of client $i$’s ownership revocation, two distinct scenarios emerge.

1. First, if remaining owners exist, the group key for the other owners is updated.
   - The cloud storage generates a group key, denoted as $G_K_{N+1}$, by encrypting the results of XOR operations on $h(M)$ and session $(N + 1)$ with the cloud storage’s $SK_C$.
   - The cloud storage generates a distribution key $RGK$, designed for the remaining owners. The $RGK$ created in this process is referred to as $RGK_{other}$.
   - The cloud storage sends $RGK_{other}$ to the CS to request it to be sent to the data owner.

2. Second, if there are no remaining owners, all information stored in the cloud storage and the CS at $h(M)$ is deleted.
   - The cloud storage deletes all data to optimize storage efficiency.
   - The cloud storage notifies the CS that all information regarding $h(M)'$ has been erased.

Step 6. **Send refreshed group key.** The CS sends the group key to the client based on the $ACL (h(M), r_n)$. Suppose that a client with $h(M)$ is client $j$, who is an existing owner.

1. If the CS receives $RGK_{other}$ from the cloud:
   - The CS performs an XOR operation on client $j$’s random value $r_j$ and $RGK_{other}$, and the operation result is $CRGK_j$ for client $j$.
   - The CS sends $CRGK_j$ to client $j$.
   - The client $j$ recovers $G_K_{N+1}$ from $CRGK_j$, and the client stores $G_K_{N+1}$.

2. If the CS receives a notification from the cloud storage that all information has been deleted:
   - The CS deletes all information related to $h(M)'$ stored in the $ACL$.

6. Discussion

This paper introduces a protocol enabling secure deduplication and dynamic ownership management based on secure data sharing in cloud (SeDaSC). Our proposal not only reduces reliance on the cryptographic server (CS) but also maintains high computational efficiency for clients. Additionally, it ensures safety even in scenarios involving client ownership changes. This section aims to elucidate the distinctions between our proposals and existing approaches.

6.1. Security Analysis

Certainly, the security analysis is explained with respect to the data privacy, data integrity, backward secrecy, and forward secrecy described in Section 4.2.

- **Data privacy.** In our protocol, the CS can access $M$, but $M$ has blind encryption applied to plaintext $m$. Hash functions used in blind encryption have structural safety that
cannot recover input values from hash values due to preimage resistance. Therefore, the CS cannot recover \( m \) from \( M \). The proposed protocol can ensure safety and solve key exchange problems with a relatively simple hash operation.

The data delivered to cloud storage in the proposed protocol are a value encrypted with the message-locked encryption (MLE) key by the CS. Since these data are encrypted with the same key, if they are a request for the same plaintext, they will have the same ciphertext. Therefore, cloud storage may perform deduplication on the same encrypted data.

- **Data integrity.** In our protocol, when a client uploads data, the CS verifies if the received hash value matches the one that it computes directly. The cloud storage also checks if the hash value received from the CS matches the one it calculates independently. In essence, during the upload process, data integrity is inherently confirmed, preventing the storage of corrupted data.

- **Forward secrecy.** In our protocol, when a client deletes their ownership, they are no longer included in the ownership group for that session, and they cannot access the original data. The ownership group is updated immediately when a client’s ownership changes, and the refreshed group key is also modified. Therefore, following a request for ownership deletion, whether data have been deleted or retained, the client cannot access data stored in the cloud storage.

- **Backward secrecy.** In our protocol, a client can only access data stored in the cloud if they have uploaded the data and acquired ownership. Even if a client owns the data, they do not automatically become a part of the ownership group for that session. In the proposed protocol, when a client uploads data, the ownership group is immediately updated, and the refreshed group key is changed. Therefore, even if a client uploads data, they cannot gain access to information about data previously stored by the ownership group.

Through these measures, the proposed protocol addresses the aspects of data privacy, data integrity, backward secrecy, and forward secrecy, providing a comprehensive security framework for the system.

Table 2 provides a comparison between our proposed approach and five closely related proposals from Section 2. It aims to highlight how our protocol offers enhanced security and the ability to update ownership in the context of SeDaSC. The table demonstrates the key differences and advantages of our proposal compared to existing proposals. Ref. [4] proposed the concept of convergent encryption (CE), which allows for secure data deduplication by encrypting data using the hash value of the message. However, it had the drawback of not providing a mechanism for verifying data integrity. Ref. [9] proposed a method within the framework of MLE called hash and CE with tag check (HCE2). HCE2 overcomes the limitations of CE by employing cryptographic hash functions to create tags that verify data integrity. Both of these data deduplication techniques offer privacy features but do not consider methodologies for scenarios where ownership changes. Ref. [5] proposed the SeDaSC protocol for authenticated client groups, using a CS to enhance client computational efficiency. However, it lacked data privacy as data sent to the CS were not encrypted. Moreover, it assumed the trustworthiness of the CS performing data deduplication, making it difficult to ensure data integrity since it had access to plaintext data. Ref. [7] proposed a solution to address the privacy issue in SeDaSC by using CE to provide data privacy. However, as it relies on CE, it does not guarantee data integrity. Additionally, clients had to authenticate themselves to the CS as the legitimate owners of the data to access them. And, it could lead to compliance with forward and backward secrecy. However, the purpose of this method was to prove the legitimate owner rather than manage ownership. Hence, it differs from the dynamic ownership management considered in this paper. Ref. [6] proposed a server-side deduplication protocol that considered a dynamic ownership update and was compliant with all security requirements. However, the difference lies in the fact that it does not prioritize reducing the client’s computation. Referencing Table 2, our protocol exhibits unique attributes in comparison to similar pro-
tocols, ensuring data privacy, verifying data integrity, and supporting both forward and backward secrecy concurrently. Prior existing protocols have occasionally lacked in fulfilling particular aspects or critical security requirements. In contrast, our protocol excels in delivering heightened security capabilities by comprehensively addressing these aspects.

<table>
<thead>
<tr>
<th></th>
<th>Data Privacy</th>
<th>Data Integrity</th>
<th>Forward Secrecy</th>
<th>Backward Secrecy</th>
</tr>
</thead>
<tbody>
<tr>
<td>[9]</td>
<td>O</td>
<td>O</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>[5]</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>[7]</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Ours</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

6.2. Performance Analysis

Table 3 provides a comparative analysis of the proposed protocol concerning client computational complexity and the presence of server-aided features, particularly the server computational complexity required for the dynamic ownership update. It is important to note that our proposed protocol is a type of server-side deduplication. In Section 2.1, we compare our protocol with other server-side deduplication protocols.

The computational complexity of the client refers to the amount of computational resources required when uploading or downloading data. The server computational complexity refers to the amount of computational resources required when cloud storage updates dynamic ownership. Notably, refs. [11,12] and the data download of [15] did not provide explicit mathematical formulations in the paper.

<table>
<thead>
<tr>
<th></th>
<th>Client Computational Complexity</th>
<th>Server-Aided</th>
<th>Server Computational Complexity</th>
<th>Dynamic Ownership Update</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initial Upload</td>
<td>Subsequent Upload</td>
<td>Download</td>
<td></td>
</tr>
<tr>
<td>[4]</td>
<td>$1H + 1SE$</td>
<td>$1H + 1SE$</td>
<td>$1SD$</td>
<td>X</td>
</tr>
<tr>
<td>[8]</td>
<td>$2H + 2SE + 2M + 3E$</td>
<td>$2H + 2SE + 2M + 3E$</td>
<td>$2SD$</td>
<td>O</td>
</tr>
<tr>
<td>[6]</td>
<td>$2H + 1SE + 1\oplus$</td>
<td>$2H + 1SE + 1\oplus$</td>
<td>$2H + 2SD + 1\oplus$</td>
<td>X</td>
</tr>
<tr>
<td>[12]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>[13]</td>
<td>$1H + 5M + (5 + 2B) * E + 1DDH$</td>
<td>$1H + 5M + (5 + 2B) * E + 1DDH$</td>
<td>$1H + 1KDF + 3M$</td>
<td>O</td>
</tr>
<tr>
<td>[14]</td>
<td>$3H + 1SE + 1M + 1E + 1DS$</td>
<td>$3H + 1SE + 1M + 1E + 1DS$</td>
<td>$1H + 1SD$</td>
<td>O</td>
</tr>
<tr>
<td>[7]</td>
<td>$1H + 1SE$</td>
<td>$1H + 1SE$</td>
<td>$1SD$</td>
<td>O</td>
</tr>
<tr>
<td>[15]</td>
<td>$1H + 1DS + 1PRE-Dn + 1PRE-En + 1SE$</td>
<td>$1H + 1DS + 1PRE-Dn + 1PRE-En$</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Ours</td>
<td>$2H + 2\oplus$</td>
<td>$2H + 2\oplus$</td>
<td>$1\oplus$</td>
<td>O</td>
</tr>
</tbody>
</table>

O: offer; X: not offer; SE: symmetric key encryption; SD: symmetric key decryption; H: hash function; DS: digital signature; DDH: solving the decisional Diffie–Hellman (DDH) problem; KDF: key derivation function; B: data block size; \(\oplus\): XOR operation; M: multiplication; E: exponentiation, PRE-Dn: proxy re-encryption Dn function, PRE-En: proxy re-encryption En function, PRE-ReEn: proxy re-encryption ReEn function.
• Client computational complexity:

1. **Upload.** Among the existing server-side deduplication protocols, [4,7] stand out in terms of minimal client computational requirements for data uploads, as depicted in Table 3. These protocols involve a single \( H \) and a single \( SE \) during both initial and subsequent uploads. In contrast, our protocol necessitates two operations of \( H \) and two operations of \( \oplus \). \( SE \) algorithms are typically resource-intensive and computationally complex, used for encrypting data. On the other hand, operations like transforming an input into a fixed-size hash value using \( H \) and relatively simpler bit-wise operations like \( \oplus \) are generally performed faster and more efficiently. However, actual performance may vary based on factors such as the algorithm used, implementation methods, and hardware configurations, among others.

2. **Download.** Even for data download, our protocol outperforms [4,7] since they require a single \( SD \) while our protocol requires only a single \( L \). Table 3 reveals that our approach excels in minimizing computational complexity when uploading and downloading data, offering an efficient solution for clients.

• Server computational complexity, on the other hand, pertains to the computational resources required by the cloud storage server when re-encrypting data and distributing group keys during ownership changes. The inclusion of dynamic ownership update features in [6,15] and our approach is a distinct advantage. These features guarantee data confidentiality during ownership transitions and prevent departing clients from accessing data still stored in the cloud storage.

• Server-aided capabilities, a feature supported by [6‐8,11,13,14] and our approach, refer to actions taken to obtain message-independent encryption keys through a key server. Our protocol generates MLE keys through independent key servers.

6.3. Analytical Synthesis

The proposed secure deduplication protocol is constructed based on Ail et al.’s SeDaSC protocol [5], aimed at improving client computational efficiency while incorporating the principles of Hur et al.’s dynamic ownership update [6]. Our proposed protocol shares similarities with the SeDaSC protocol but introduces the following key differences:

• Mitigation of CS dependency: The CS, which encrypts data on behalf of clients, allows clients to significantly reduce their computational workload. However, due to the CS’s capability to access data in plaintext, it must be completely trusted. Our proposal employs blind encryption on plaintext to prevent unauthorized access to plaintext by the CS. As a result, our protocol offers a secure solution, particularly in environments where trust in CS security is low.

• Dynamic ownership update: When applying data deduplication in cloud storage environments, situations arise where ownership information changes. Two common scenarios involve either the original data owner modifying or deleting their data, resulting in the revocation of ownership, or a new client uploading data that match existing data, granting them ownership rights. Such ownership changes can occur frequently in cloud storage services and must be appropriately managed to ensure the security of the service. To solve this issue, our protocol involves the cloud storage storing data in an encrypted format using a secret key. When a change in ownership occurs, the data are re-encrypted with a new key. This newly generated key is then distributed to clients by the CS. This approach enables the prevention of revoked clients from accessing data and ensures that newly added clients cannot access previously uploaded data.

• Client computational efficiency: SeDaSC optimizes client computational efficiency by assigning encryption operations to the CS. However, to generate the same ciphertext for identical data, sending plaintext to the CS is necessary, demanding trust in the CS. To address this, we implemented blind encryption on plaintext, preventing CS access to plaintext. In our proposal, blind encryption requires one hash function
and one XOR operation. Table 3 highlights our protocol’s superior computational efficiency on the client side. While our proposal involves more computations than SeDaSC, it still demonstrates greater efficiency than previously proposed server-side deduplication protocols.

This paper introduces a protocol that extends upon the SeDaSC protocol, emphasizing enhanced safety measures and diverse security aspects. Specifically, our protocol mitigates data privacy concerns by minimizing dependency on the CS, enhances storage efficiency through data deduplication techniques, and incorporates dynamic ownership management functionality. Since our protocol is based on SeDaSC, the primary point of comparison is the SeDaSC protocol. Therefore, Table 4 provides a comparison between the SeDaSC protocol and ours concerning computational and communication overhead. In this paper, the variable $\lambda$ denotes the size of the data. Specifically, in Table 4, $\lambda_P$ represents the size of the secret key, and $\lambda_C$ signifies the size of the ciphertext. These variables were utilized for the analysis of communication overhead as presented in Table 4. Firstly, our protocol marginally increases the client’s computational load while reducing dependency on the CS. However, it is noteworthy that the overall communication overhead remains unchanged. Secondly, for dynamic ownership management, interaction among the client, CS, and cloud is necessary. The ownership update mentioned in Table 4 occurs when the ownership group of data changes due to another client. The majority of this interaction involves XOR operations, which minimizes the burden on each entity due to the XOR operations’ computational efficiency. Consequently, our protocol addresses security concerns present in SeDaSC while maintaining similarity in computational and communication overhead aspects.

Table 4. Comparison of overheads between the SeDaSC protocol and our protocol.

<table>
<thead>
<tr>
<th></th>
<th>SeDaSC Protocol</th>
<th>Our Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Computational</td>
<td>Communication</td>
</tr>
<tr>
<td></td>
<td>Overhead</td>
<td>Overhead</td>
</tr>
<tr>
<td><strong>Upload</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client</td>
<td>0</td>
<td>$\lambda_P$</td>
</tr>
<tr>
<td>CS</td>
<td>$RBG + 1H + 1\oplus + 1SE$</td>
<td>$\lambda_K + \lambda_C$</td>
</tr>
<tr>
<td>Cloud storage</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Download</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client</td>
<td>0</td>
<td>$\lambda_K$</td>
</tr>
<tr>
<td>CS</td>
<td>$1SD$</td>
<td>$\lambda_P$</td>
</tr>
<tr>
<td>Cloud storage</td>
<td>0</td>
<td>$\lambda_C$</td>
</tr>
<tr>
<td><strong>Ownership update</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CS</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Cloud storage</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

$H$: hash function; $\oplus$: XOR operation; $RBG$: random bit generation; $SE$: symmetric key encryption; $SD$: symmetric key decryption; $\lambda_P$: the size of plaintext; $\lambda_H$: the output size of hash function; $\lambda_C$: the size of ciphertext; $\lambda_K$: the size of secret key; $N_C$: the number of clients in group.

7. Conclusions

This paper proposes an efficient and secure data deduplication protocol based on the secure data sharing in cloud (SeDaSC) protocol. Our proposal addresses three key aspects of SeDaSC. First, our approach enhances data privacy. In SeDaSC, the cryptographic server (CS) performs complex encryption operations on the behalf of clients, potentially compromising data privacy as the CS can access plaintext. Our approach mitigates this concern by having clients perform blind encryption using hash values of plaintext, thus preventing information exposure. Second, our approach enhances the efficient utilization of cloud storage space. SeDaSC introduced a data-sharing method using cloud storage. However, it lacks an effective method for storage space management. Data deduplication technology stands out as an efficient approach to managing data, involving the prevention of storing data twice in cloud storage if they are already uploaded. This technique has
the potential to save storage space by up to 90% [3]. Our proposed protocol enables data deduplication based on client ownership stored in an access control list (ACL), thereby ensuring efficient use of cloud storage space. Third, our approach ensures the secure management of data, including dynamic ownership management, when the composition of a group changes. SeDaSC does not consider specific scenarios related to the addition, modification, or deletion of client ownership, which may lead to security issues. While SeDaSC only allows access to stored data by clients with legitimate ownership, it does not address scenarios involving ownership changes. For instance, security issues might arise when newly registered clients are granted access to previous data or when revoked ownership clients continue to access data stored in cloud storage. Our proposed protocol addresses these concerns by allowing for the registration or deletion of ownership and distributing refreshed group keys for the respective session to prevent both previous and new clients from accessing ciphertext.

The SeDaSC protocol requires only one hash function operation, thereby providing advantages in terms of client computation cost and time during data uploads. However, this approach may pose data privacy concerns as it exposes plaintext to the CS. Our protocol addresses this issue by conducting one hash function and one XOR operation. As indicated in Section 6.2 and Table 3, our protocol exhibits significantly less computational complexity on the client side compared to previously proposed server-side deduplication protocols. Hence, our protocol offers increased safety compared to SeDaSC and reduces computational cost and time in contrast to other protocols.

Through this research, we have presented a protocol that maintains client computational efficiency while supporting data privacy, data integrity, and ownership updates. These improvements are expected to contribute to secure and efficient data management in cloud storage environments. Cloud storage serves primary purposes: a service provider securely stores data from clients while delivering a service, or a client directly stores data in platforms such as Google Drive, AWS S3, and others. Our proposed protocol holds the advantage of guaranteeing computational efficiency for the client. As a result, our protocol has advantages in scenarios where clients directly encrypt and store their data on cloud storage. For future work, it would be interesting to incorporate efficient key revocation techniques [24] into our proposed protocol. This integration would lead to a substantial enhancement in the ownership update.

Author Contributions: Conceptualization, M.L.; methodology, M.L.; validation, M.L. and M.S.; formal analysis, M.L.; investigation, M.L.; writing—original draft preparation, M.L.; writing—review and editing, M.L. and M.S.; supervision, M.S.; project administration, M.S.; funding acquisition, M.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Basic Research Program through the National Research Foundation of Korea (NRF) funded by the MSIT (grant number: 2021R1A4A502890711).


Conflicts of Interest: The authors declare no conflict of interest.

References


Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.