energies

MDPI

Article

Locked Deduplication of Encrypted Data to Counter Identification Attacks in Cloud Storage Platforms

Taek-Young Youn [1,*], Nam-Su Jho [1,*], Keonwoo Kim [1], Ku-Young Chang [1] and Ki-Woong Park [2,*]

[1] Electronics and Telecommunications Research Institute (ETRI), 218 Gajeong-ro, Yuseong-gu, Daejeon 34128, Korea; wootopian@etri.re.kr (K.K.); jang1090@etri.re.kr (K.-Y.C.)

[2] Department of Computer and Information Security, Sejong University, 209 NeungDong-Ro, Gwangjin-Gu, Seoul 05006, Korea

* Correspondence: taekyoung@etri.re.kr (T.-Y.Y.); nsjho@etri.re.kr (N.-S.J.); woongbak@sejong.ac.kr (K.-W.P.); Tel.: +82-42-860-0622 (T.-Y.Y.); +82-42-860-1860 (N.-S.J.); +82-2-6935-2453 (K.-W.P.)

Received: 25 March 2020; Accepted: 20 May 2020; Published: 29 May 2020

Abstract: Deduplication of encrypted data is a significant function for both the privacy of stored data and efficient storage management. Several deduplication techniques have been designed to provide improved security or efficiency. In this study, we focus on the client-side deduplication technique, which has more advantages than the server-side deduplication technique, particularly in communication overhead, owing to conditional data transmissions. From a security perspective, poison, dictionary, and identification attacks are considered as threats against client-side deduplication. Unfortunately, in contrast to other attacks, identification attacks and the corresponding countermeasures have not been studied in depth. In identification attacks, an adversary tries to identify the existence of a specific file. Identification attacks should be countered because adversaries can use the attacks to break the privacy of the data owner. Therefore, in the literature, some counter-based countermeasures have been proposed as temporary remedies for such attacks. In this paper, we present an analysis of the security features of deduplication techniques against identification attacks and show that the lack of security of the techniques can be eliminated by providing uncertainness to the conditional responses in the deduplication protocol, which are based on the existence of files. We also present a concrete countermeasure, called the time-locked deduplication technique, which can provide uncertainness to the conditional responses by withholding the operation of the deduplication functionality until a predefined time. An additional cost for locking is incurred only when the file to be stored does not already exist in the server's storage. Therefore, our technique can improve the security of client-side deduplication against identification attacks at almost the same cost as existing techniques, except in the case of files uploaded for the first time.

Keywords: data storage systems; encrypted storage; encryption; identification attack; privacy

1. Introduction

Currently, data owners' privacy is one of the fundamental qualities required of information services, including storage services where users entrust their data to a remote storage server [1]. To guarantee the privacy of stored data, ciphertexts (or encrypted files) instead of original files are stored in remote storages. For the efficiency of storage services, deduplication is a very important functionality, since the cost of storage management can be reduced by eliminating duplicated files [2]. However, determining whether two original files are the same or not from viewing two ciphertexts is not easy. To support the deduplication of ciphertexts, *message-locked encryption* was proposed [3]. This technique helps users to generate the same ciphertext for the same file and allows users to enjoy the advantages

of the deduplication of ciphertexts. After the basic concept of deduplication of encrypted data has been introduced, value-added techniques have been proposed [4–6]. For example, in [4], block-level deduplication has been studied to support the deduplication functionality for partially duplicated files. In [5], an dedicated deduplication scheme has been introduced for cloud media center where naive deduplication of encrypted data cannot be applied. In [6], the search of near-duplicated data for encrypted data has been studied.

There are two types of deduplication techniques: server-side and client-side. In server-side deduplication, clients upload their data regardless of whether the files already exist and the server periodically eliminates duplicated copies. In client-side deduplication, the client can upload files without actually sending them to the server. When the same file already exists in the server's storage, the server verifies the client's ownership of the file and assigns the ownership without actually receiving the file from the client. Server-side deduplication techniques can reduce storage costs; however, they cannot improve the cost of communication. In contrast, client-side deduplication techniques can reduce the cost of not only storage but also of communication because the server and the client can omit the actual transmission of a file when it is already stored in the server's storage. In this study, we focus on client-side deduplication, which has more advantages in terms of efficiency than server-side deduplication. Although client-side deduplication can improve the performance of a storage service by eliminating the transmission and storage of repeated data, there are some security concerns that should be resolved [7]. In the literature, poison, dictionary, and identification attacks are considered as threats against client-side deduplication techniques.

Thus far, several techniques have been designed to provide deduplication that includes security against known attacks [2,8–13]. Known attacks can be classified into the three following types.

- Poison attacks. In this attack, an adversary tries to store maliciously manipulated data instead of the original data such that legitimate users obtain the harmed data. The poison attack is the most serious threat to storage services with deduplication because the purpose of an adversary is to store poisoned data instead of the original data. If the server has a poisoned file, clients could lose their data. A more serious problem is that the clients can be infected with malicious codes injected by the adversary into the poisoned file. Hence, security against poison attacks should be guaranteed.
- Dictionary attacks. In this attack, an adversary tries to guess a file by constructing a dictionary of possible data and then tests the correctness of all files in the dictionary. For the correctness test, an adversary can mount an offline dictionary attack where the test can be performed either offline or online, which permits the adversary to check a guessed file by performing an online protocol. An adversary can test a number of guessed files in offline attacks only if they can obtain data stored in a storage server, and, thus, the security against offline attacks are not considered as a real threat to security. Although an adversary can mount an online dictionary attack easily, we have a simple countermeasure against it that limits the number of protocol executions. Note that, in general, private data such as employment contracts are the typical target of dictionary attacks.
- Identification attacks. In this attack, an adversary tries to identify the existence of a specific file. In general, the adversary can check the existence of such file by using conditional protocol executions in client-side deduplication techniques. The objective of identification attacks is to break the privacy of clients based on the measured feature. Although dictionary attacks seem similar to identification attacks in that an adversary makes a set of guessed data and checks the existence of each guess, they are different attacks that should be dealt with separately. The main difference between these two attacks is that dictionary attacks use the existence of a guessed file to check the correctness of the guessed file, whereas identification attacks use the existence of a known file to break the privacy of a client-side deduplication technique. Note that, for security algorithms or protocols, the aim is to prevent any possibility that can cause a security weakness. Identification attacks are considered for the security of storage services because these attacks show the possibility of breaking the file owners' privacy.

There exist sufficiently useful tools for securing deduplication techniques, except against identification attacks. If the storage server verifies the correctness of stored encrypted data by evaluating their hash value, security against the poison attacks can be easily guaranteed because the hash value of a corrupt file cannot be identical to that of the original file. This technique is used in most deduplication techniques because of its simplicity and effectiveness [2,3,7–14]. Most of the existing works were conducted to focus on the security against dictionary attacks. To counter dictionary attacks, one can adopt a well-designed server-aided technique, where a key server helps users to generate a deterministic secret key for a file [14]. No adversary can perform (offline) dictionary attacks because he/she needs to access the key server for each guess. It is desirable to rely less on a trusted third party because of the cost of maintaining such an entity. In [15], the first single-server scheme was proposed, which supports cross-user deduplication without any additional independent servers. To achieve the objective, each client should perform a costly cryptographic operation for executing a password-authenticated key exchange (PAKE) protocol to upload a file. In [16], an improved PAKE-based construction was proposed with better security by adopting a re-encryption technique. The scheme in [16] guarantees that a compromised client cannot learn whether or not a certain file has already been uploaded by someone else. All schemes in [15,16] guarantee the asserted security features by performing an upload procedure so that the existence of the same file does not get revealed to anyone. Note that, clients have to send their files to the storage server in [15,16]. All the schemes in [15,16] lose one of the main merits of client-side deduplication, namely, the reduction in the communication cost of transmitting a file.

In contrast to other attacks, identification attacks have been addressed by only a few research studies. An adversary can mount an identification attack using the conditional response returned by the storage server, which depends on whether the file already exists in the storage [7]. In other words, a client-side deduplication technique is not secure against an identification attack if an adversary can determine the existence of a file in the storage through the response of the storage server (To mount identification attacks, an adversary needs to continuously query the existence of a file to identify the first client who uploads it. Such a malicious behavior can be easily detected by storage server since continuous protocol fails is not a normal pattern. To detect and filter out an adversary who uses service provider as an oracle, we generally counter the number of oracle-calls until pre-defined threshold. However, such attacks cannot be fundamentally blocked since an adversary can generate and use multiple accounts or a number of adversaries collude to mount identification attacks as in online dictionary attacks. Hence, it is reasonable to assume an adversary who continuously queries a file). Unfortunately, these conditional responses cannot be eliminated because the storage server should notify the client of the existence of a file if the file is already stored in the server's storage. Thus far, approaches have been proposed to reduce the lack of security by providing randomness to the conditional response. In existing techniques, to prevent identification attacks, the storage server selects a secret counter, ctr, and then performs a client-side deduplication as if the file in question did not exist until ctr number of users had uploaded the same file.

The implementation of this countermeasure for identification attacks is considerably simple and efficient in practice; however, it has a limitation in that only the server can select the level of security. In existing techniques, a higher level of security can be expected when the counter is large; however, the cost of the deduplication is higher because the deduplication functionality does not operate until ctr number of users have uploaded the same file, where ctr is the predefined counter. Therefore, the clients may expect the server to provide a larger counter for the sake of greater security; however, the server may want to use a smaller counter to provide better efficiency. Unfortunately, as mentioned earlier, only the server can control the size of the counter in existing techniques. Moreover, a client does not know the size of the counter because such information is private to the server. Hence, there is no guarantee that the server will choose a sufficiently large counter. In general, the storage server is trusted and assumed to be an honest but curious party, which implies that the server can behave

maliciously if it can obtain an advantage by doing so without being detected by users. In this case, existing countermeasures cannot provide sufficient security against an honest but curious server.

In this paper, we revisit the security aspect of deduplication techniques against identification attacks and show that the lack of security can be eliminated by providing uncertainness to the conditional responses returned by the storage server. We also present a specific countermeasure, called the time-locked deduplication technique, which is based on a time-based mechanism that locks some information for a predefined time and releases it only after the time has been reached. Recall that the common disadvantage of existing techniques is that only the storage server can determine the security level and clients cannot verify this level. Our countermeasure constitutes the first attempt to design a method that can allow users to select the level of security. The proposed technique requires additional cost for the functions that lock and unlock stored information; however, these functions are performed only if the file to be stored does not exist in the server's storage. Hence, the slight increase in computational cost does not affect the efficiency of the deduplication services.

2. Uncertainness for Conditional Responses

Before describing our protocol, we briefly review the basic procedure of client-side deduplication techniques and the identification attacks against these techniques.

2.1. Client-Side Deduplication

We now focus on the simple client-side deduplication procedure for encrypted data. Note that we do not explain the detailed procedure for generating encrypted data using a message-locked encryption scheme (To support the deduplication functionality for encrypted data, a class of encryption schemes has been proposed, the so-called message-locked encryptions, which use a data-derived encryption key to guarantee that any user can generate the same ciphertext for a file) to simplify the explanation. To upload a ciphertext C, a client *Clt* and a server *Srv* perform the following steps:

- Step 1. To verify the existence of the ciphertext C, *Clt* generates a tag tag_C for the file. In general, a cryptographic hash function $h(\cdot)$ is used to generate the tag, and the tag is simply computed as $\mathsf{tag}_C = h(C)$. Then, *Clt* sends the tag tag_C to *Srv*.
- Step 2. *Srv* verifies the existence of C using the given tag_C. If the file is not stored in its storage, *Srv* sends N/A to *Clt* and allows the client to upload the file; otherwise, *Srv* generates a challenge and sends it to *Clt*.
- Step 3. If *Clt* receives N/A from *Srv*, he/she uploads the ciphertext C, stops the protocol, and deletes the ciphertext C from his/her storage; otherwise, *Clt* computes π as a proof to answer a given challenge by using the ciphertext C. Then, *Clt* sends π to *Srv*.
- Step 4. *Srv* verifies the given proof π. If it is correct, *Srv* affirms *Clt*'s ownership of the ciphertext C; otherwise, *Srv* sends N/A to *Clt*.
- Step 5. When *Clt* receives N/A, *Clt* uploads the ciphertext C and deletes it from his/her storage; otherwise, *Clt* deletes the ciphertext C from his/her storage without uploading the file.

As seen in the aforementioned procedure and in Figure 1, the client *Clt* uploads a ciphertext C to the storage server *Srv* only if the file does not already exist in the server's storage, which also means that the client and the server can complete the file upload procedure very efficiently when the file is already stored in the server's storage. Note that, in Figure 1, the PoW protocol refers to the protocol that is used to test whether the client actually has the file or not by performing an interactive protocol.

Recall that the client uploads a file only if the same file does not already exist in the server's storage, and the conditional protocol execution is one of the features of client-side deduplication techniques, as compared with server-side deduplication techniques. An adversary can use the conditional behavior to break the security of client-side deduplication services in terms of users' privacy because the property of the behavior can be used to determine the existence of a file. An adversary can test whether or not a

file is stored in the server's storage, and, thus, he/she can determine the identity of the client who uploads a file for the first time by checking the existence of the file before and after the client uploads it.

As we have discussed here, the source of the lack of security of client-side deduplication techniques against identification attacks is the conditional responses because they provide a clue to the existence of files. Hence, we focus on a method that can provide uncertainness to the conditional responses.

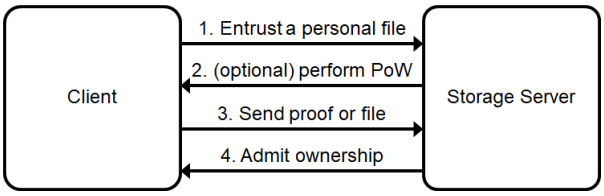

Figure 1. Client-side deduplication.

2.2. Privacy Leakage in Client-Side Deduplication

Before discussing the privacy leakage in client-side deduplication in detail, let us recall that a checking step exists. When a user initiates a protocol execution to upload a ciphertext C, he/she evaluates a tag from the file and gives the tag to the service provider. Here, the tag is used for identifying a specific ciphertext C, and, thus, a hash value of the file is generally used as a tag because of the collision resistance of the underlying hash function.

The conditional protocol execution cannot be removed in client-side deduplication because a file will be stored only if the same file does not exist in the server's storage. Any client can determine whether a file exists in the server's storage or not by running the client-side deduplication protocol because the server's response to the client is based on the existence of the ciphertext C. The binary response returned by the server reveals the existence of a file, which can help adversaries perform identification attacks. Note that the feature can be used only once because the file is stored after the protocol execution and the status of a file does not change after a valid client has uploaded the file. However, the client who first uploads the file is still vulnerable to identification attacks because the adversary can test the existence of a file several times until a legitimate client uploads the file. The adversary can query the existence of the file without changing the existence status of the file. A simple strategy of the adversary is to suspend the protocol execution in the final step without uploading the file. Thus, the adversary can test the existence of a file several times before a client uploads the file. By testing the existence of a file before and after a target user uploads the file, an adversary can breach the privacy of the user who owns the file.

The privacy of users or stored data is one of the important security concerns in data storage services, and, thus, the identification attack is a significant security lacuna for client-side deduplication techniques, as indicated in [7]. Moreover, the attack can be a serious threat to storage services if a file containing unique or rare data is chosen as a target file.

2.3. Binary Decision $\neq$ Binary Information

As explained in the previous section, the weakness of client-side deduplication techniques is basically caused by the conditional response of the server. Because the response is determined by the existence of a file, the server's response gives binary information to an adversary. We can improve the security of client-side deduplication techniques by providing uncertainness to the binary response. In other words, client-side deduplication techniques can be secure against an adversary if the adversary cannot determine the existence of a file using the server's response. The server's response is either *YES* or *NO*. If the file exists in the server's storage, the server will return a *YES* response; otherwise, it will return a *NO* response. Now, we can consider the provision of uncertainness to each response.

2.3.1. Providing Uncertainness to a YES Response

In the *YES* response case, we cannot provide uncertainness to the server's response. The client may delete his/her file without uploading it to the server when the server already possesses the file. In other words, the client will delete the file when the server returns a response. Hence, if we provide uncertainness to a *YES* response, the client could lose his/her file because the file does not actually exist in the server's storage, even though the server returns a *YES* response. The loss of clients' data is a serious problem that cannot be allowed to occur in storage services. Therefore, it is not possible to provide uncertainness to the *YES* response.

2.3.2. Providing Uncertainness to a NO Response

Recall that a *NO* response means that the server does not have the file queried by the client. In contrast to the *YES* response, the *NO* response can be provided with uncertainness by returning it even if the file exists in the server's storage. When the *NO* response is fake, the client loses the opportunity to complete the upload procedure at a lower cost; however, the storage service remains effective.

2.4. Conditionally Activated Deduplication System

As analyzed above, the security of client-side deduplication techniques can be improved by providing uncertainness to the server's response, in particular to the *NO* response. The next step is to design an effective technique for providing uncertainness. To achieve this purpose, the server should be able to return a fake *NO* message, even if the target file is actually stored in its storage. Note that, to provide security against identification attacks, the detection of a faked response should be difficult. Thus, the objective of this study is to design an efficient mechanism that can support the server to create an undetectable fake *NO* message.

The main idea of our solution is to lock stored data such that they cannot be deduplicated until certain predefined conditions are met. In our solution, encrypted data and certain related information are stored in an incomplete form and then are completed once the predefined conditions are met so that they can be used for deduplication. Throughout this paper, we use two terms, namely, lock and unlock, to identify the procedures for storing incomplete data and recovering complete data from these incomplete data, respectively. Similarly, we call our techniques that use lock and unlock functions as locked deduplication.

Here, we briefly review the count-locked system, which is used in the literature for locked deduplication. The first locked deduplication mechanism can be implemented by maintaining a counter for the number of queries entered by clients. Existing techniques are designed using a counter-based mechanism because its implementation is relatively easy. Stated simply, the server chooses a counter *ctr* for a file that is stored for the first time and does not use the stored file for deduplication until *ctr* clients store the same file. In this case, the number of queries is the condition that should be satisfied to unlock stored data. Note that, if the threshold number is revealed to an adversary, he/she can increase the success probability of an attack by counting the number of queries entered for the target file. Using this knowledge, the adversary can also consume a predefined number of queries. In other words, the adversary can ask a number of queries to increase the counter until it is equal to the threshold number. To prevent adversarial attempts, one should use a privately selected and maintained number or an arbitrary random number as a counter. As mentioned, counter-based techniques are easy to implement, and, thus, the design of existing techniques is based on this feature. However, a disadvantage of the techniques is that only the server can control the security by choosing the counter. We can expect a higher level of security if the server chooses a larger counter because the level of randomness increases together with the size of the counter; however, this results in an increase in the cost on the server side. If the server does not want to incur an additional cost for improving clients' privacy, it may choose a smaller counter.

3. Time-Based Locked Deduplication Technique Using a Pairing Operation

We begin this section by discussing some of the requirements for designing a suitable scheme. Then, we describe the basic idea of our time-based locked deduplication technique and provide a concrete scheme that is based on a pairing operation defined over elliptic curves. We also provide an improved scheme that guarantees greater security against dictionary attacks compared with the basic construction. We want to remark that the word *lock* means that the functionality of deduplication is disabled by some mechanism.

3.1. Design Considerations

Before describing our construction, we discuss some problems that should be considered in the design of an effective scheme. Note that the only difference between locked and ordinary deduplication is the functionality of the locking and unlocking of stored data, and, thus, we discuss them in detail.

3.1.1. System Model

In the proposed deduplication system, three parties work as follows:

- Client. This party outsources data to a cloud storage. Before uploading his/her file, the client can lock the file so that it cannot be used in deduplication until a predefined time. The file can be unlocked with the help of the time server after the time when the locked file can be unlocked.
- Cloud Storage Server. This party provides data storage services to users and uses a deduplication mechanism to save cost for supporting services. The server may behave maliciously to save costs if the malicious behavior does not get detected. During the deduplication process, if the file to be stored already exists in the server's storage, the server can perform a proof-of-ownership protocol with the client to verify that the client really owns the file.
- Time Server. This party generates and publishes time-related information for each time period. We assume that the time server does not collude with a storage server.

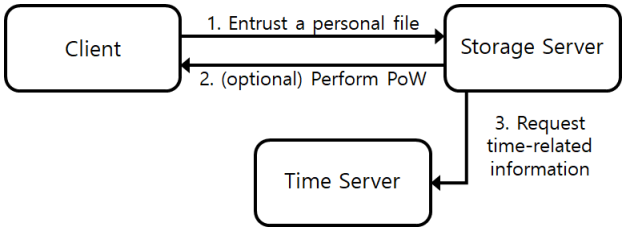

Figure 2. Time-based Locked Deduplication System.

Note that, as seen in Figure 2, our system requires a time server, which incurs an additional cost for storage services unlike a counter-based approach. However, the extra cost permits us to support a higher level of security that cannot be supported by existing techniques. In this paper, we first propose a novel countermeasure against identification attacks that is based on a time-locked approach, and the novelty is the main contribution of this study.

3.1.2. Who Locks the Information?

In existing locked deduplication techniques, only the server can determine the condition for unlocking a locked file. Measures to improve the privacy of users inevitably increase the cost of the system. Hence, it is desirable to permit the client to choose the level of privacy because the server gains no advantage by increasing the level. In existing counter-based locked deduplication techniques, only the server can determine the security level and clients cannot measure the security strength of the service, and, thus, the level of security of the existing countermeasures against identification attacks cannot be guaranteed. To guarantee the security against identification attacks, we need a way to prove

the security level of the applied countermeasure or a new technique that can allow users to choose a sufficient security level.

3.1.3. Who Unlocks the Locked Information?

When a tag is unlocked, the privacy protection provided by the time-based lock mechanism is removed. Hence, the party that can unlock a tag should be reliable in the sense that it does not unlock stored data intentionally for malicious purposes. In existing counter-based locked deduplication techniques, the server can unlock any locked information without being detected to reduce the cost of managing its databases. Hence, it is not desirable to permit the server to unlock any locked data. Then, the ability to unlock data could be given to the client or to an additional (reliable) party.

3.1.4. Requirements

As for other security schemes, the requirements for a useful locked deduplication system can be classified into two branches: functionality and security. The primary functional requirement is correctness. In an examination of the correctness of the system, the following aspects should be checked.

- Ordinary functionalities for client-side deduplication services should operate correctly, regardless of the additional functionalities for managing locked information. For this feature, (1) a file uploaded for the first time should be stored in the server's storage, (2) a client can obtain the ownership of a file without uploading it if the file is already stored in the server's storage, and (3) a user can retrieve data if the ownership of the file is assigned to the user.
- A locked deduplication mechanism should operate correctly in the sense that (1) locked information should not be unlocked until a predefined time t is reached and (2) the locked information must be unlocked after t is reached.

As mentioned repeatedly, the manner in which the proposed technique supports the use of locked information is different from that of ordinary client-side deduplication systems, and, thus, two additional functions for locking and unlocking are required. For the technique to become practical, the first function, namely, lock, should be easy to operate for all users who generate locked information, and the second function, namely, unlock, should be difficult to operate for all users until the predefined time is reached.

3.2. Basic Idea: Time-Based Locked Deduplication

Here, we describe the idea of our time-locked tag mechanism, where locked information can be unlocked after a predefined time. To prevent an adversary from unlocking locked information before the predefined time, one needs somewhat complex techniques. A possible time-lock technique is to impose the computational cost on the unlocking. In the literature, various cryptographic techniques have been reported whose designs are based on a cost-based time-lock mechanism. The well-known Bitcoin system is a good example of a cost-based mechanism. In the system, a hash value can be considered as the locked information and a pre-image of the hash value is the corresponding unlocked information. In other words, to unlock locked information, the system must find a pre-image of a hash value by performing an exhaustive search, which requires a considerable amount of time. A disadvantage of cost-based time-lock systems is the inaccuracy of the time for unlocking because locked information can be unlocked more easily by a computer with a higher capability. This inaccuracy is not desirable in the case of our system because an exact time for unlocking is required. An additional type of time-locked tag mechanism can be designed by adopting a time server. The role of this server is to publish information for unlocking locked information. Stated simply, the system can be implemented using a hash function. In this example, a (random) value is locked by applying a hash function. The party that locked the value entrusts the random value to a trusted server together with

the time when it should be published. Then, the server publishes the entrusted value at the planned time. In practice, service providers are assumed to be honest but curious entities that behave honestly unless they can obtain merit from a malicious behavior without being detected. In this case, this simple example cannot support a secure time-based lock because the server can abuse its ability by giving entrusted values to any malicious party and its behavior cannot be easily detected. We present a specific technique in the following section.

3.3. Basic Construction: Naive Locked Deduplication Scheme

Now, we describe our scheme in detail. Let *Clt* be a client, and let *Srv* be a server that supports storage services. Let Srv_{time} be the time server that helps the storage server to unlock locked information. In other words, Srv_{time} helps the storage server to unlock any stored locked information after the predefined time of unlocking has passed. In our construction, we use the following algorithms and system parameters. Let $KGen(F)$ be the key-generation function that evaluates a key from the file F, $Enc_k(F)$ be the encryption function that encrypts the file F using the key k, $TGen(F)$ be the tag-generation function that returns a tag for the file F, and $prng(\ell, sd)$ be a pseudo-random number generator (prng) that generates an ℓ-bit random number from the seed sd. Let $\mathcal{E}$ be an elliptic curve, and let P be a point on the elliptic curve such that $\langle P \rangle = \mathcal{E}$. Let e be the bilinear pairing operation defined over the elliptic curve $\mathcal{E}$. Let s be a private key of Srv_{time}, and let $Q = sP$ be a public key of the time server. Let H be a hash function that maps a string of arbitrary length to a point on $\mathcal{E}$.

Our scheme comprises two phases for controlling locked information. The first one is the online phase, which is similar to ordinary client-side deduplication, and the second one is the offline phase, which is aimed to eliminate locked information when a predefined time has elapsed.

3.3.1. Online Phase

Before describing the scheme in detail, we emphasize that the server *Srv* maintains its storage as if it had two storage units *DB* and DB_L, which are the storage units for ordinary and locked data, respectively. In *DB*, the server maintains $\{Tag_C, C, UID_C\}$ for a ciphertext C, where C is the encryption of a file F, Tag_C is the tag for the ciphertext C, and UID_C is the list of users who store C in the server. In DB_L, the server maintains $\{t, R, Tag_C, LC, uid\}$ for the user whose identity is *uid*, where t and R are time-related information for unlocking and LC is the locked ciphertext.

- Step 1. To check the existence of the file F, the client *Clt* computes the following:

$$k = KGen(F), C = Enc_k(F), \text{ and } Tag_C = TGen(C).$$

  Then, the client sends the tag Tag_C to *Srv*.
- Step 2. The server *Srv* searches for a file in its storage using the given tag. If there is a file that is indexed by the tag, *Srv* assigns the ownership of the searched file to the client and notifies *Clt* of its existence; otherwise, the server notifies *Clt* that the file is absent by sending N/A to *Clt* and allows the client to upload the file.
- Step 3. If *Clt* receives N/A from *Srv*, the client chooses a random r and a time t, and computes $h_t = H(t)$, $R = rP$, and $sd = e(h_t, Q)^r$. Then, he/she generates a locked ciphertext as

$$LC = Lock(rs, C) \tag{1}$$

  where $rs = prng(\ell, sd)$. Then, the client uploads the time-locked information

$$\{t, R, Tag_C, LC, uid_F\} \tag{2}$$

  to the server and deletes the file F from his/her storage; otherwise, *Clt* deletes the file F without uploading it.

3.3.2. Offline Phase

Note that the main objective of the offline phase is to unlock stored locked information, whose predefined unlock time has been reached. For this phase, we need the help of the time server Srv_{time}, which returns $sH(t)$ for each time t, where s is the private key of Srv_{time}, as defined above. The offline phase is performed periodically, and, thus, a set of data are treated at once. However, to simplify the explanation, we focus on the case where a single file is processed.

- Step 1. For each time t, Srv_{time} publishes the time-related value $T_s = sH(t)$. The time server can transmit time-related values to storage servers by various means; however, we consider the case where the time server publishes the information on a bulletin board. Then, at time t, Srv_{time} may publish $T_s = sH(t)$ on its bulletin board.
- Step 2. When the time-related value T_s is published by Srv_{time}, the server Srv retrieves the locked information $\{t, R, Tag_C, LC\}$ from its storage and computes

$$sd = e(T_s, R) \text{ and } rs = prng(\ell, sd). \tag{3}$$

Then, the storage server can unlock the time-locked ciphertext LC by computing

$$C = Unlock(rs, LC). \tag{4}$$

When the file indexed by Tag_C is first unlocked, the server verifies the correctness of the tag by comparing Tag_C with $TGen(C)$. If the two values are the same, the storage server Srv deletes $\{t, R, Tag_C, LC, uid\}$ from DB_L, sets $UID_C = \{uid\}$, and stores $\{Tag_C, C, UID_C\}$ to DB. When the file indexed by Tag_C already exists in the server's storage, the server tests whether the unlocked ciphertext is identical to the existing one and adds uid to UID_C if the equality test holds. If the equality test does not hold, the server regards the user whose identity is uid as a malicious user and adds the user's identity to a black list (If one tag can be used for indexing two or more files, we have to perform the equality test procedure more carefully. However, in this paper, we assume that only one file can be identified by one tag for simplicity of explanation; the assumption is reasonable because of the collision resistance of the hash function that is widely used as a tag generation function. Moreover, we can guarantee the assumption by using a longer hash value as a tag). Regardless of the results of the equality test, the server deletes $\{t, R, Tag_C, LC, uid\}$.

In the description of the proposed scheme, we used $Lock(\cdot)$ and $UnLock(\cdot)$ without providing a detailed description because the functionalities of locking and unlocking can be implemented by various means. A simple candidate method is to use a prng. In this case, we generate an ℓ-bit random sequence for a given input seed sd, where ℓ is the bit size of the ciphertext, and the random sequence is used as a random mask for the ciphertext C:

$$LC = C \oplus rs, \tag{5}$$

where $\oplus$ is the bitwise XOR operation. If we use a symmetric encryption scheme to randomize the ciphertext, ℓ is the size of the key for the encryption scheme. In this case, the locked ciphertext can be computed as

$$LC = Enc_{rs}(C). \tag{6}$$

If we use a prng as $Lock(\cdot)$, the unlock function involves generating the same random sequence and removing the random mask. When we use an encryption scheme as $Lock(\cdot)$, the unlock function involves decrypting the locked ciphertext.

Remark 1. *Note that, in the second step of the offline phase, the server deletes a set of locked information indexed by* Tag_C *without verifying its correctness by evaluating the tag from the unlocked file when the file indexed by*

the tag already exists in the server's storage. In this case, the server can efficiently verify the correctness of the unlocked file by simply comparing it with the stored file.

3.4. Secure Locked Deduplication Scheme

In the previous section, we described the basic idea of our scheme's construction. In this section, we present an improved version of our scheme, which supports greater security than the basic scheme. The main difference is the security against dictionary attacks. Although the main interest in this study is the security against the identification attacks, we will show that our approach and existing key-server based approaches can be used at the same time to provide greater security against dictionary attacks and identification attacks. To achieve the objective, unlike in the basic construction, in the secure version, we use a key server Srv_{key} that helps users to generate a secret key for a file. Please refer to [14] for details regarding the key server.

The offline phase of the improved scheme is identical to that of the basic scheme, and, thus, we describe only the online phase of the stronger locked deduplication technique as follows.

Online Phase

- Step 1. First, the client *Clt* contacts the key server Srv_{key} to generate the key k for the file F. Then, the client computes the following:

$$C = Enc_k(F) \text{ and } Tag_C = TGen(C).$$

  To check the existence of the file F, the client sends the tag Tag_C to *Srv*.
- Step 2. The server *Srv* searches for the file in its storage using the given tag. If the file does not exist in the storage, the server notifies *Clt* that the file is absent by sending N/A and allows the client to upload the file; otherwise, the server and the client perform the proof-of-ownership protocol to prove that the client really owns the file. If the client passes the test, *Srv* assigns the ownership of the searched file to the client and informs *Clt* that the file exists. If the client cannot pass the test, the server terminates the protocol execution.
- Step 3. If *Clt* receives N/A from *Srv*, the client chooses a random r and a time t and computes $h_t = H(t)$, $R = rP$, and $sd = e(h_t, Q)^r$. Then, he/she generates a locked ciphertext as

$$LC = Lock(rs, C) \tag{7}$$

  where $rs = prng(\ell, sd)$. Then, the client uploads the time-locked information

$$\{t, R, Tag_C, LC, uid_F\} \tag{8}$$

  to the server and deletes the file F from his/her storage; otherwise, *Clt* deletes the file F without uploading it.

In the second step, a proof-of-ownership technique for proving the client's ownership of the file is performed. The server knows the valid tag for C, although it does not have the ciphertext, which could be a weak point in the proposed scheme because a malicious server could pretend to have an unlocked ciphertext and ordinary proof-of-ownership techniques cannot guarantee that a server has a file. Hence, we cannot use existing proof-of-ownership techniques and, therefore, we present a new technique in Section 3.5 for our constructed scheme.

3.5. Proof of Ownership

Recall that one of the merits of our scheme is that a client can choose the security level against identification attacks by determining the time a file can be unlocked. Because of this feature, the service provider should maintain a set of locked files until they are unlocked even if the files are generated

from the same file. For a greedy server, it can increase the cost for supporting storage services. Moreover, when the unlocked file does not exist in the server's storage, the server should pay the cost for receiving a locked file from a client even if a set of locked files for the same data are already stored in its storage. If possible, the server may try to reduce the operational expenses or it may be tempted to pretend to possess the unlocked file when a locked file is planned to be unlocked shortly. In this case, the server can eliminate duplicated (but stored in a different form) copies and acknowledge receipt of the same files.

To prevent the server from behaving as described previously, for our scheme, we need a new proof-of-ownership technique that can permit users to check whether the server actually has the unlocked file or not because the server can easily pretend to have the unlocked file, although it only has the locked files (The existence of a file can be tested based on a fixed tag, and thus, it is easy for the server to pretend to have unlocked files without in fact having them). When the server can pretend to have the unlocked files, some clients may not be able to retrieve their data if the (stored) locked files are not unlocked until they want to retrieve their data. Hence, we have to enable users to determine whether the server actually has the unlocked file. For this reason, we present a scheme that permits the server to prove a user's ownership only if the server has the unlocked file.

For the design of our scheme, we modified a well-known proof-of-ownership technique proposed by Halevi et al. in [17]. In the following procedure, we assume that the client has confirmed the existence of the target file using the tag Tag_C. If the server has the unlocked information of the ciphertext C, it maintains

$$\{Tag_C, \pi_C\}, \tag{9}$$

where π_C is the last block of C.

- Step 1. *Srv* generates a challenge and sends it to *Clt* with π_C. Here, the challenge is a set of random indices of the leaf nodes of a Merkle tree for the ciphertext C.
- Step 2. For a given challenge, *Clt* compares π_C with the last block of C. If the two values are identical, *Clt* accepts the given challenge as valid and responds to it by generating the sibling paths for all the indices presented as the challenge and sends them to *Srv*; otherwise, *Clt* rejects the challenge and terminates the protocol execution.
- Step 3. Based on the correctness of the given sibling paths, *Srv* determines *Clt*'s ownership of the ciphertext C.

We need the proposed proof-of-ownership technique to prevent the server's malicious behavior by checking whether *Srv* has an unlocked file or not. Until (at least) one file is unlocked, the server cannot generate a valid challenge without some partial information of the unlocked ciphertext C, and, thus, a user can be convinced that the server actually possesses the file C. In the above construction, the server cannot obtain the last block of the unlocked ciphertext from a locked ciphertext, and, thus, we can use the last block of the ciphertext as a proof. If we are required to deal with the case where the server can obtain the last block of any unlocked ciphertext despite not having any unlocked data, a stronger solution can be adopted in the above procedure (In the literature, many proof of ownership schemes have been proposed [17–22]. Amongst these, we use Halevi et al.'s scheme in the design of our new scheme because of its simplicity. However, we can use other techniques to provide stronger security or improved performance if needed). The client may use a proof-of-ownership technique instead of receiving the last block of the unlocked ciphertext. In this case, the client may choose to send a challenge to the server and the server should respond to the challenge by generating a valid proof.

4. Analysis

In this section, we discuss the security features and computational complexity of the proposed locked deduplication technique. The qualities required of our scheme can be summarized as follows:

- *Correct Deduplication*

1. Repeated copies can be deduplicated without incurring the loss of the original file.
2. When predefined conditions hold, the storage server can open a set of locked files and perform a deduplication of repeated files.

- *Secure Deduplication*

1. (For poison attacks) No adversary can break the security of the storage service system by uploading a corrupted file.
2. (For dictionary attacks) An adversary cannot obtain any information to mount either offline or online guessing attacks without any limitation in the number of key generations.
3. (For identification attacks) The storage server can only deduplicate unlocked files.
4. (For identification attacks) Locked files can be unlocked only if the predefined conditions hold, and only the authorized and trusted party (or a set of parties) can control the conditions.

It is easy to verify the correctness of the proposed scheme. In our scheme, the server performs a deduplication of unlocked files based on their tags as in ordinary deduplication techniques, which implies that repeated copies can be correctly deduplicated after unlocking. The other part to consider is the correctness of the unlocking process. The time server is designed to publish $T_s = sH(t)$ at time t, and the published value can be used for computing the random information rs. A stored locked file can be unlocked using the random value rs, and, thus, any locked information can be correctly unlocked after a predefined time.

4.1. Security Analysis

The next problem is the security of the proposed technique. First, we show the security of the proposed locked deduplication technique against poison attacks as follows.

Security Feature 1. The proposed locked deduplication technique is secure against poison attacks if the underlying hash function is collision resistant.

Proof. Recall that, in poison attacks, malicious users may attempt to upload a corrupt file with the ultimate aim of swapping valid files stored by other users with corrupt files in the offline deduplication phase. Note that, in the online storing phase, locked files are sent to the server for storage. Therefore, simply storing corrupt files in the locked form cannot affect the files of other users. Our scheme can resist this threat by verifying the correctness of a tag in the offline deduplication phase. In this phase, the storage server evaluates the hash of an unlocked file and compares it with the stored tag. Through the verification of the correctness of the tag, the security of the proposed scheme against the uploading of corrupt files can be guaranteed as no adversary can store a file with a hash value that is identical to that of the original (non-corrupt) file because of the collision resistance of the underlying hash function. □

Recall that susceptibility to dictionary attacks is an inevitable weakness of deduplication services because of their use of message-locked encryption. We also note that dictionary attacks can be classified into two types of guessing attacks: offline and online. An adversary cannot mount an offline guessing attack if the storage server does not reveal the stored files. Because the service provider is assumed to be an honest entity, we can trust that it will not reveal any stored files. Hence, we can focus on the security of the proposed scheme against online guessing attacks, where an adversary attempts to correctly guess a file in online protocol executions. Note that the purpose of a countermeasure against online guessing attacks is to convince the adversary that it is difficult to verify guessed information.

Security Feature 2. The proposed locked deduplication technique with countermeasures (also called Locked-Dedup$^+$ in Figure 1) supports stronger security against online guessing attacks in that an adversary's capability to mount attacks can be controlled.

Proof. To mount an online guessing attack for a file, an adversary needs the key of the file, which can be obtained by performing an interactive protocol with the key server. However, by adopting a key server that generates a key for a file on behalf of the user, we can limit the adversary's ability to obtain the encryption key for the file. As a result, as in [14], we can thwart the adversary's ability to mount online guessing attacks by limiting the number of key generations. □

The final security feature that we will demonstrate is the security against identification attacks. Because we provide the feature of the locked deduplication functionality, the security against identification attacks can be verified by showing that the functionality correctly prevents the server from maliciously identifying a stored file without the help of an authorized party, such as the time server. The features can be verified as follows:

Security Feature 3. The proposed locked deduplication technique supports greater security against identification attacks in the sense that the lock of the deduplication functionality operates correctly. Specifically, the proposed technique guarantees that

- (1) The storage server can perform the deduplication operation only for unlocked files; and
- (2) The storage server cannot unlock any locked file without the help of authorized parties.

Proof. The first security requirement is that the storage server can perform the deduplication operation for unlocked files. Recall that the storage server will behave maliciously only if it can obtain an advantage from such a behavior and if its malicious behavior will not be detected by clients. In our technique, the server can check the existence of a file in its storage even if the file is not unlocked. However, the server cannot perform deduplication using a locked file because it requires checking that Tag_C is properly computed from the corresponding ciphertext C. Moreover, the server only has the locked file LC. The embedded encryption algorithm guarantees that the non-negligible information of the plaintext cannot be revealed from the ciphertext. Therefore, the server needs to unlock LC first before checking the validity of Tag_C.

The next part of the proof is to check that the storage server cannot unlock any locked files. To prove this, we need to assume that the embedded encryption algorithm and the prng are secure and the following cryptographic assumption.

Definition 1. *Bilinear Diffie-Hellman (BDH) Assumption: Let P be a generator of a cyclic group with a bilinear map $e(\cdot,\cdot)$, and let a, b, c be randomly chosen integers. The BDH assumption means that it is infeasible to compute $e(P,P)^{abc}$ with the input (P, aP, bP, cP).*

Note that the encryption key is $rs = prng(\ell, sd)$. From the properties of the encryption algorithm and the cryptographic prng, unlocking LC is equivalent to obtaining the secret key rs and it is also equivalent to obtaining the seed for the prng sd itself. Assume that there is a polynomial-time algorithm Adv that can compute the value of sd without any help from the time server. In other words,

$$Adv(P, h_t = H(t), R = rP, Q = sP) = e(h_t, P)^{rs} = sd.$$

This implies that $Adv(\cdot)$ can solve the BDH problem as $Adv(P, aP, bP, cP) = e(aP, P)^{bc} = e(P,P)^{abc}$.

Therefore, from the BDH assumption, there exists no polynomial-time algorithm Adv that can compute sd with public inputs for the server. This concludes the proof. The storage server cannot unlock a file without any help from the time server. □

4.2. Efficiency

In terms of computational complexity, the proposed deduplication technique is almost identical to existing deduplication techniques, although an additional cost is incurred by providing the additional functionality called locked deduplication. Our technique requires a slightly higher cost than ordinary

deduplication schemes to provide the functionality of locked deduplication. However, the additional cost is incurred only for files stored for the first time. The cost of the upload is identical to that in the ordinary deduplication techniques if the file is already stored in the server's storage, which means that the performance of the proposed technique is almost identical to that of ordinary deduplication, except in a few cases. Now, we will analyze the efficiency of the proposed scheme against identification attacks by comparing it with existing methods.

To compare the computational cost and the communication overhead of the proposed technique with those of existing techniques, we present them in Figures 3–5, and we also compare their performances in Tables 1 and 2 by counting the number of operations. In the comparison of efficiency, we will focus on the cost of the countermeasure against identification attacks because it is the main contribution of this work. Hence, we will compare our work with the counter-based solution proposed in [7] because it is the only known technique against identification attacks. To examine the scale of the additional cost for countering identification attacks, we also include the naive client-side deduplication in the comparison. Therefore, we will compare three techniques: the naive client-side deduplication, the counter-based client-side deduplication, and our scheme. We also summarize their features in Table 3.

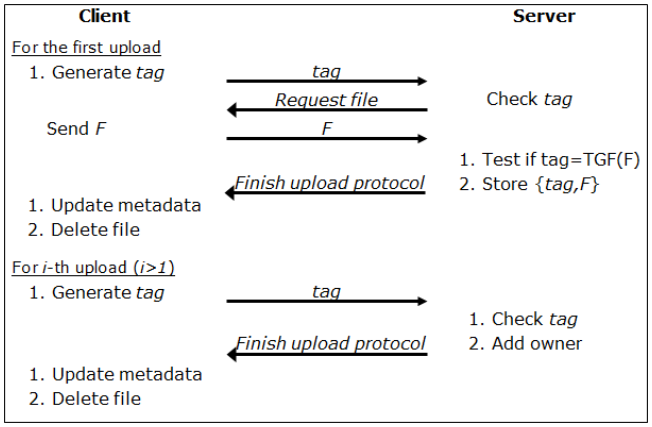

Figure 3. Naive deduplication.

As can be seen in Figure 3, in the naive client-side deduplication, a client can finish an upload protocol execution without actually transmitting his/her file when the same file is already maintained by the server. Therefore, except for the first user who should actually transmit his/her file, we can reduce the cost of communication by eliminating the data transmission step. In the counter-based scheme, as seen in Figure 4, the server chooses a random threshold so that the functionality of client-side deduplication is disabled until the counter is increased by one for each upload request and it is less than the predefined random threshold. One merit of the counter-based approach is the efficiency of the functionality because the only additional procedure is to set a threshold and to maintain a counter for a file. However, the scheme is not designed to support encrypted data and, therefore, we cannot use it for encrypted data. To be sure, we can modify it to support encrypted data by using one of the message-locked encryption techniques. However, the scheme cannot support client-side deduplication for encrypted data in the current form. In terms of the security against identification attacks, the scheme is secure if the storage server willingly consumes its resource because a large counter may guarantee a higher level of security than a small one. For example, if a curious server uses a considerably small number or a fixed number for the threshold, we cannot expect sufficient security. Because a storage server is regarded as an honest but curious adversary, we need a way to force the server to support sufficient security against identification attacks. In Figure 5, we illustrate the proposed technique. Because we already provided a detailed explanation for it in Section 3, we have not provided any further explanation for the figure. Unlike the two other schemes, the proposed scheme supports encrypted data for deduplication, and, thus, there is a step for encryption in Figure 5.

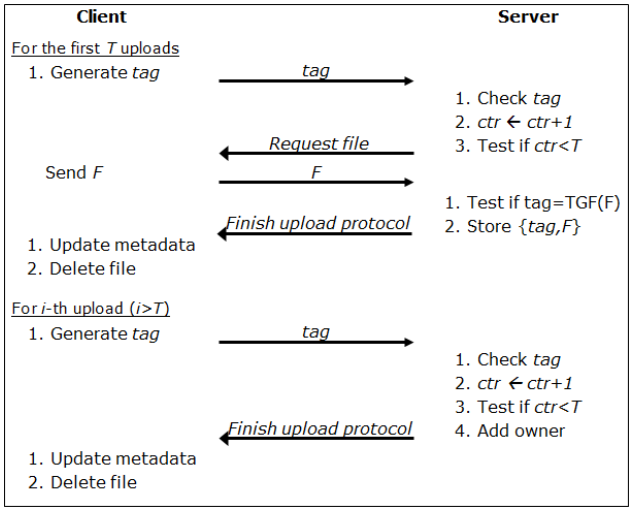

Figure 4. Counter-based client-side deduplication.

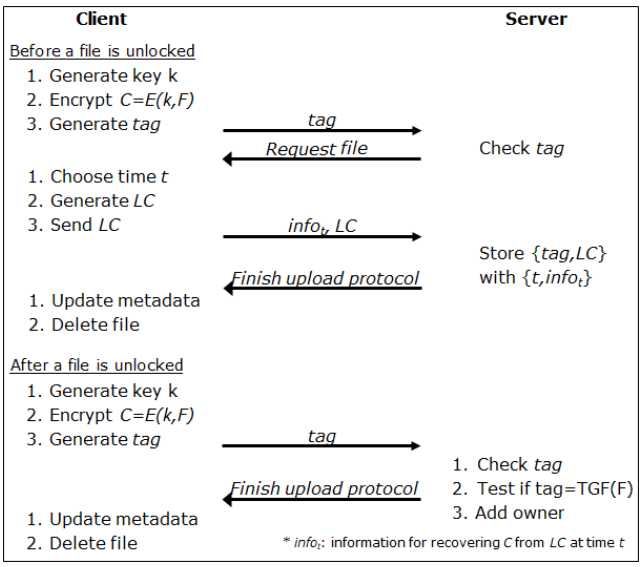

Figure 5. Time-locked deduplication (Basic Construction).

In Tables 1–3, we use the following notations. C_K: cost of key generation, C_{Tg}: cost of tag generation, C_{Tv}: cost of tag verification (in general, $C_{Tg} = C_{Tv}$.), C_E: cost of encryption for full data, C_S: cost of testing the existence of a file, C_L: cost of locking a file, ℓ_T: length of tag, ℓ_F: length of (encrypted) file to be stored, ℓ_t: length of $info_t$. For fair cost evaluation, we compare the basic model (in Section 3.3) with other schemes to focus on the cost of the countermeasure against the identification attacks and we consider the full model (in Sections 3.4 and 3.5) to figure out all security features of our scheme.

Table 1. Cost of the upload procedure for locked files.

	Computational Cost		Size of Message		# of Round
	Server	**Client**	**Server**	**Client**	
Naive CS-Dedup	$C_S + C_{Tv}$	C_{Tg}	-	$\ell_T + \ell_F$	4
Counter-based CS-Dedup [7]	$C_S + C_{Tv}$	C_{Tg}	-	$\ell_T + \ell_F$	4
Locked-Dedup (Basic Model)	C_S	$C_K + C_E + C_{Tg} + C_L$	-	$\ell_T + \ell_F + \ell_t$	4

Table 2. Cost of the upload procedure for unlocked files.

	Condition for Unlock	Computational Cost		Size of Message		# of Round
		Server	Client	Server	Client	
Naive CS-Dedup	From the second uploader	C_S	$C_K + C_E + C_{Tg}$	-	ℓ_T	2
Counter-based CS-Dedup [7]	From the T-th uploader	C_S	$C_K + C_E + C_{Tg}$	-	ℓ_T	2
Locked-Dedup (Basic Model)	After the time $T_{threshold}$	C_S	$C_K + C_E + C_{Tg}$	-	ℓ_T	2

Table 3. Properties.

	Encrypted Data	Poison Attacks	Identification Attacks	Dictionary Attacks	Supporting PoWs
Naive CS-Dedup	X	O (Can be prevented by performing server-side tag verification)	X	X (Known methods can be applied to encrypted data)	Any PoWs schemes can be used
Counter-based CS-Dedup [7]	X		Δ		
Locked-Dedup (Full Model)	O		O	O (Section 3.4)	Proposed PoW (Section 3.5)

As summarized in Tables 1 and 2, the counter-based countermeasure [7] and our techniques guarantee the same performance for unlocked files because they work similar to the naive client-side deduplication technique. Therefore, almost all features are the same for unlocked files. The main difference is the cost of operations for locked files. In the naive scheme, only the first uploader performs a costly upload procedure owing to the absence of the same file. In the counter-based scheme, the first T uploaders perform a costly upload procedure because the server does not support client-side deduplication until T copies have been uploaded. Note that, in most of the existing schemes, the server should verify a given tag for the first uploaded files to counter poison attacks. However, in our scheme, the server does not verify a given tag in the upload procedure. The correctness of a tag is verified in an offline manner. Therefore, one of the important merits of our scheme is the online efficiency of the upload procedure for locked files. Note that, for server-based services, it is necessary to reduce the cost of operations for the servers because they have to support costly functions for a number of clients. In this case, our scheme outperforms other existing schemes. To be sure, clients must compute costlier operations than other schemes for additional functionalities; however, the computational capability of devices is sufficient to support the operations, which can be computed on-line, thereby indicating that the performance of the upload procedure can be improved by precomputing the operations. Moreover, among additional operations, two operations are also required for other schemes to support encrypted data, as in our scheme. If we focus on the security against identification attacks, only the cost of locking a file is the additionally required burden for the additional security. Although one more message is required for our scheme, its size is negligibly short compared with the file size. Therefore, unlike the number of operations in the table, our scheme can support a more efficient upload process in the online step than other schemes. Particularly, the cost of the online step is significantly efficient for the server, which should perform a number of upload queries made by its clients. As shown in Table 3, we summarize the important features of client-side deduplication techniques. As can be seen in the table, only the proposed scheme can support all security features. We mark with a Δ the security of the scheme in [7] against identification attacks because it is secure against these attacks in some sense; however, the security of this is relatively lower than that of our scheme. As seen in Table 3, other schemes are not designed to support encrypted data nor can they support the security against dictionary attacks because the existing countermeasures can only be used for encrypted data. Note that the way of generating an encryption key is the key point strategy in existing techniques,

and the countermeasures cannot be used for the naive client-side deduplication and the counter-based countermeasure [7] to counter dictionary attacks.

Note that the existing counter-based countermeasure presented in [7] can be applied to any client-side deduplication techniques, and the performance of the resultant scheme is almost identical to that of the underlying client-side deduplication technique. Before the predefined counter is reached, the client-side deduplication includes a counter-based countermeasure that operates as if there were no duplicate copy, which means that the scheme waives the advantages of client-side deduplication until a counter meets a predefined value to increase the security against identification attacks. It is not easy to compare the performance of this counter-based countermeasure with that of our technique because the former waives the advantage of client-side deduplication until a predefined counter is reached, whereas our technique waives the advantage until a predefined time is reached. The main difference between the two is that our scheme allows clients to choose the security level for their data, whereas the counter-based technique cannot guarantee the security level supplied by the storage server. We also emphasize that clients cannot measure the security level provided by the storage server, which means that, because an honest but curious adversary can behave maliciously if its malicious behavior is not detected, a malicious server can use a counter-based countermeasure with a lower level of security.

So far, the protocol-level overhead analysis in terms of the computational cost and the communication overhead has been presented. This study has focused on the novel protocol design as a proof of concept rather than on the performance improvement regarding the deployment into a real cloud platform. Considering a real deployment of our proposed scheme into a real cloud platform, a more rigorous and practical comparison of the performance of the propose scheme versus others, considering the system-level I/O parallelism and memory caching, could be an important task to go one step forward to the completeness of the proposed scheme.

5. Conclusions

Deduplication is a well-known and useful technique for efficient storage services; however, we must solve some security problems that arise from its useful functionality. Among various attacks against deduplication techniques, we focused on identification attacks that should be countered but are not fully secured. Because identification attacks can weaken the privacy of storage services, it is meaningful to study the attacks and propose a suitable countermeasure.

In this paper, we revisited deduplication techniques against identification attacks and showed that the attacks can be countered by providing uncertainness to the conditional responses in the deduplication protocol. As a specific countermeasure, we proposed a time-locked deduplication technique that locks the deduplication functionality until a user-selected time has elapsed. The proposed technique can improve the security of client-side deduplication against identification attacks at almost the same cost as existing techniques, except in the case of files uploaded for the first time.

As the next step of this study, we are exploring more advanced security-aware deduplication protocol which will be undertaken on a real cloud computing service. Therefore, a more rigorous comparison of the performance of the propose scheme versus others could be an important task to improve the completeness of the proposed scheme. Consequently, we set the more rigorous performance evaluations as our further work.

Author Contributions: Conceptualization and methodology, T.-Y.Y.; validation, N.-S.J., K.K., K.-Y.C., and K.-W.P.; formal analysis, T.-Y.Y. and N.-S.J.; investigation, K.K. and K.-Y.C.; writing—original draft preparation, T.-Y.Y.; writing—review and editing, N.-S.J., K.K., K.-Y.C., and K.-W.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean government [20ZR1300, Development of core technologies for trust data connectome to realize a safe data society]. Also, this work was supported by the National Research Foundation of Korea (NRF) (NRF-2020R1A2C4002737) and the Institute for Information and Communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-00420).

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

References

1. Lytras, M.D.; Raghavan, V.; Damiani, E. Big data and data analytics research: From metaphors to value space for collective wisdom in human decision making and smart machines. *Int. J. Semant. Web Inf. Syst. IJSWIS* **2017**, *13*, 1–10. [CrossRef]
2. Storer, M.W.; Greenan, K.; Long, D.D.E.; Miller, E.L. Secure data deduplication. In *4th ACM International Workshop on Storage Security and Survivability*; ACM: New York, NY, USA, 2008; pp. 1–10.
3. Bellare, M.; Keelveedhi, S.; Ristenpart, T. Message-locked encryption and secure deduplication. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 2013.
4. Chen, R.; Mu, Y.; Yang, G.; Guo, F. BL-MLE: Block-Level Message-Locked Encryption for Secure Large File Deduplication. *IEEE Trans. Inf. Forensics Secur.* **2015**, *10*, 2643–2652. [CrossRef]
5. Zheng, Y.; Yuan, X.; Wang, X.; Jiang, J.; Wang, C.; Gui, X. Toward Encrypted Cloud Media Center with Secure Deduplication. *IEEE Trans. Multimed.* **2017**, *19*, 251–265.
6. Cui, H.; Yuan, X.; Zheng, Y.; Wang, C. Enabling secure and effective near-duplicate detection over encrypted in-network storage. In *IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications*; IEEE: New York, NY, USA, 2016; pp. 1–9.
7. Harnik, D.; Pinkas, B.; Shulman-Peleg, A. Side channels in cloud services: Deduplication in cloud storage. *IEEE Secur. Priv. Mag.* **2010**, *8*, 40–47. [CrossRef]
8. Li, J.; Chen, X.; Li, M.; Li, J.; Lee, P.P.C.; Lou, W. Secure deduplication with efficient and reliable convergent key management. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 1615–1625. [CrossRef]
9. Li, J.; Chen, X.; Xhafa, F.; Barolli, L. Secure deduplication storage systems with keyword search. In *Proceedings of the AINA 2014*; IEEE: New York, NY, USA, 2014; pp. 971–977.
10. Li, J.; Li, Y.K.; Chen, X.; Lee, P.P.C.; Lou, W. A hybrid cloud approach for secure authorized deduplication. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *26*, 1206–1216. [CrossRef]
11. Marques, L.; Costa, C. Secure deduplication on mobile devices. In *Proceedings of the 2011 Workshop on Open Source and Design of Communication*; ACM: New York, NY, USA, 2011; pp. 19–26.
12. Shin, Y.; Kim, K. Efficient and secure file deduplication in cloud storage. *IEICE Trans. Inf. Syst.* **2014**, *E97-D*, 184–197. [CrossRef]
13. Xu, J.; Chang, E.C.; Zhou, J. Weak Leakage-Resilient client-side deduplication of encrypted data in cloud storage. In *Proceedings of the ASIA-CCS 2013*; ACM: New York, NY, USA, 2013; pp. 195–206.
14. Bellare, M.; Keelveedhi, S.; Ristenpart, T. DupLESS: Server-aided encryption for deduplicated storage. In *Proceedings of the 22nd USENIX Conference on Security*; USENIX Association: Washington, DC, USA, 2013; pp. 179–194.
15. Liu, J.; Asokan, N.; Pinkas, B. Secure Deduplication of Encrypted Data without Additional Independent Servers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*; ACM: New York, NY, USA, 2015; pp. 874–885.
16. Liu, J.; Duan, L.; Li, Y.; Asokan, N. Secure Deduplication of Encrypted Data: Refined Model and New Constructions. In *Proceedings of CT-RSA 2018*; Springer: Cham, Switzerland, 2018; pp. 374–393.
17. Halevi, S.; Harnik, D.; Pinkas, B.; Shulman-Peleg, A. Proofs of ownership in remote storage systems. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*; ACM Press: New York, NY, USA, 2011; pp. 491–500.
18. Blasco, J.; di Pietro, R.; Orfila, A.; Sorniotti, A. A tunable proof of ownership scheme for deduplication using bloom filters. In *Proceedings of the CNS 2014*; IEEE: New York, NY, USA, 2014; pp. 481–489.
19. Husain, M.I.; Ko, S.Y.; Uurtamo, S.; Rudra, A.; Sridhar, R. Bidirectional data verification for cloud storage. *J. Netw. Comput. Appl.* **2014**, *45*, 96–107. [CrossRef]
20. di Pietro, R.; Sorniotti, A. Boosting efficiency and security in proof of ownership for deduplication. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*; ACM: New York, NY, USA, 2012; pp. 81–82.

21. Xu, J.; Zhou, J. Leakage resilient proofs of ownership in cloud storage. In *International Conference on Applied Cryptography and Network Security*; Springer: Cham, Switzerland, 2014; pp. 97–115.
22. Yu, C.-M.; Chen, C.-Y.; Chao, H.-C. Proof of ownership in deduplicated cloud storage with mobile device efficiency. *IEEE Netw.* **2015**, *29*, 51–55. [CrossRef]