Client-Side Deduplication to Enhance Security and Reduce Communication Costs

Keonwoo Kim, Taek-Young Youn, Nam-Su Jho, and Ku-Young Chang

Message-locked encryption (MLE) is a widespread cryptographic primitive that enables the deduplication of encrypted data stored within the cloud. Practical client-side contributions of MLE, however, are vulnerable to a poison attack, and server-side MLE schemes require large bandwidth consumption. In this paper, we propose a new client-side secure deduplication method that prevents a poison attack, reduces the amount of traffic to be transmitted over a network, and requires fewer cryptographic operations to execute the protocol. The proposed primitive was analyzed in terms of security, communication costs, and computational requirements. We also compared our proposal with existing MLE schemes.

Keywords: Deduplication, Security, Poison attack, Duplicate-faking attack, Erasure attack, Communication, Computation, MLE.

I. Introduction

Data deduplication is a technique for eliminating redundant copies of duplicate data without storing the same data as a way to save storage space and network bandwidth. Recently, the need for secure deduplication of encrypted data between the client and storage server is increasing because clients who use storage services do not want to expose the data stored in storage providers to malicious users, [1]–[7]. In 2013, Bellare and others presented Message-locked encryption (MLE) [1], which provides encryption schemes to achieve secure deduplication in cloud storage environments. MLE schemes include convergent encryption (CE), hash and CE without a tag check (HCE1), hash and CE with a tag check (HCE2), and randomized convergent encryption (RCE). All of these techniques are variations of the basic convergent encryption idea [8]. The key used for encryption and decryption in MLE schemes is derived from a message. Thus, clients can obtain the same key and ciphertext from the same plaintext message regardless of who encrypted it. The storage server does not re-store the duplicate ciphertext if a copy of the ciphertext already exists in its storage. If the MLE is used for server-side deduplication, the clients should always upload their files to the server without consideration of any duplication of the files. If an MLE scheme is used for client-side deduplication, the clients do not need to upload duplicate copies over the network. The server will simply update the metadata which indicates an additional owner of the file.

Although client-side MLE schemes efficiently utilize network resources, they create critical security issues regarding cross-user deduplication which means that identical duplicated files from different users will be detected and removed safely [6], [9]–[10]. In particular, such schemes can be vulnerable to a poison attack, which is the most serious threat to secure deduplication. If a malicious adversary is able to replace a valid

Manuscript received Jan. 22, 2016; revised June 29, 2016; accepted Sept. 5, 2016. This work was supported by Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean government (17ZH1700, Development of storage and search technologies over encrypted database).

Keonwoo Kim (corresponding author, wootopian@etri.re.kr), Taek-Young Youn (taekyoung@etri.re.kr), Nam-Su Jho (nsjho@etri.re.kr), and Ku-Young Chang (jang1090@etri.re.kr) are with the Hyper-connected Communication Research Laboratory, ETRI, Daejeon, Rep. of Korea.

file with a poisoned file, the subsequent users will lose their original copy of the file and retrieve the corrupt version [7]. Conceptually, poison attacks include duplicate-faking attacks and erasure attacks. If a deduplication scheme is susceptible to a duplicate-faking attack, a legitimate message can be replaced by a fake message, unbeknownst to the owner. Imagine that an adversary (a) sends a tag to identify ciphertext C, and (b) uploads a poisoned ciphertext C^* instead of the non-corrupted ciphertext C. When a legitimate client attempts to upload C, enciphered from a non-corrupted plaintext M, the server determines that the tag associated with C is identical to the tag associated with C^* and thus the server does not store C. Later, a legitimate user will download C^* and decrypt it. The client expects to recover M, but in a successful duplicate-faking attack, the client recovers forged plaintext M^*. If the deduplication scheme provides security to protect against a duplicate-faking attack, the client can detect the corruption and the adversary cannot force the legitimate client to accept a corrupt message. Nevertheless, the scheme still does not prevent the deletion of the original message. Thus, the clients will no longer be able to recover their message. If the scheme can provide additional security to protect against an erasure attack, the scheme can prevent an adversary from erasing a message owned by a legitimate client. A secure deduplication scheme that prevents a duplicate-faking attack is called a tag consistency (TC) secure scheme [1]. In addition, if the scheme provides security protection against both a duplicate-faking attack and an erasure attack, the scheme is said to be strong tag consistency (STC) secure. On the other hand, a server-side CE scheme provides security against a poison attack but requires a great deal of network bandwidth.

To resolve the security problem with network resources, we propose a new secure deduplication technique to prevent a duplicate-faking attack as well as an erasure attack, and reduce the large communication bandwidth requirement. The proposed method is a fundamental primitive which can be used at file-level and block-level deduplication. The remainder of this paper is organized as follows. Section II briefly introduces an overview and provides a summary of weaknesses of the previous MLE techniques. In Section III, we describe our new proposed client-side secure deduplication scheme to protect against poison attacks. In Section IV, we analyze our primitive in terms of security, communication, and computational costs. In addition, we provide an objective evaluation of our method relative to other existing techniques.

II. Analysis of MLE Schemes and Their Weaknesses

In this section, we briefly describe four practical MLE schemes and their weaknesses.

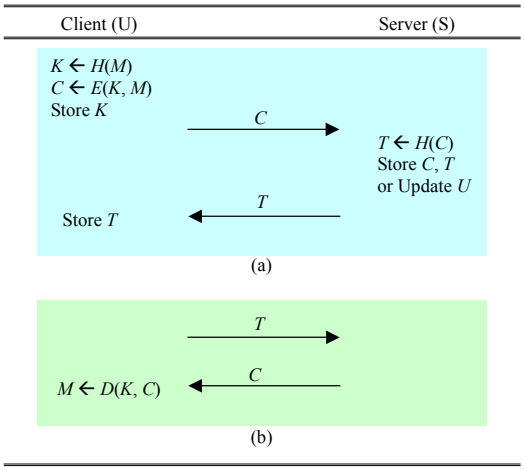


Fig. 1. CE scheme: (a) upload and (b) download protocols.

1. Convergent Encryption (CE)

In the upload protocol of the CE scheme, as shown in Fig. 1(a), a client derives $K \leftarrow H(M)$ from M and computes $C \leftarrow E(K, M)$, such that M is a plaintext message, C is a ciphertext message, K is a message-driven key, H is a cryptographic hash function, and E is a block cipher for encryption. The client then uploads C to the server and retains the value of K. Upon receipt of C, the server creates a tag $T \leftarrow H(C)$ computed using the value of C and compares T with the tags in its storage. At this time, T is used as an identifier of C to check the uniqueness of C. If no tag matches T, the server stores both C and T. Otherwise, the server will not store C; rather the server will update the meta-data to indicate that the client owns C. To sum up, CE scheme creates T on server-side and determines the duplication of C. To allow the client to retrieve C at a later time, the server returns T to the client.

Figure 1(b) shows the download protocol of the CE scheme. The client can download C by sending T, and can, therefore recover $M \leftarrow D(K, C)$ by deciphering C, using its retained K, where D is a block cipher used for decryption.

The CE scheme provides secure protection against a poison attack because the server computes a tag from the uploaded file and assures that the ciphertext file and tag are derived from the original plaintext. However, this scheme consumes significant network and bandwidth resources by sending multiple duplicate files. Moreover, the CE scheme requires larger server computational resources because it conducts hash operations on all files.

2. Hash and CE without Tag Check (HCE1)

A client in the HCE1 scheme computes a key $K \leftarrow H(M)$ from M using a hash function, H, and, derives a tag $t \leftarrow G(K)$ from M by applying a second hash function. The value of t is

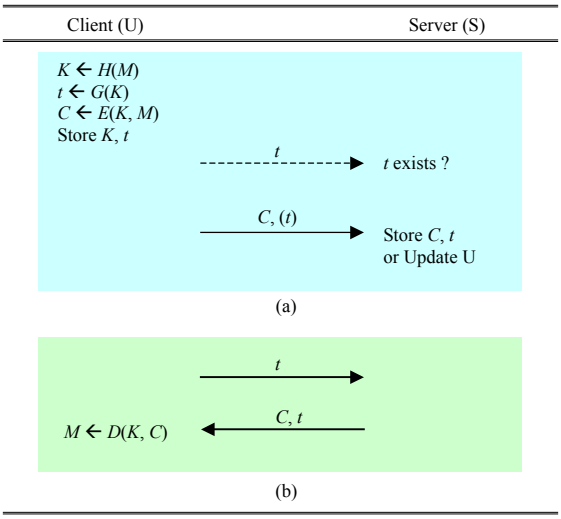


Fig. 2. HCE1 scheme: (a) upload and (b) download protocols. Dotted arrow is only applied to the client-side deduplication. A parameter in brackets is not transmitted in the client-side deduplication.

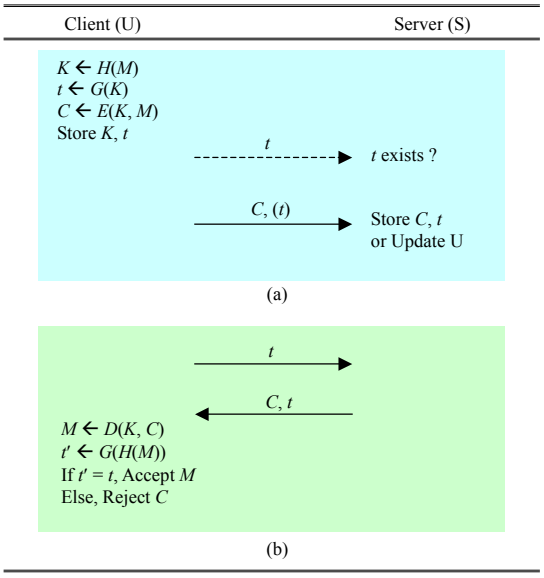


Fig. 3. HCE2 scheme: (a) upload and (b) download protocols. Dotted arrow is only applied to the client-side deduplication. A parameter in brackets is not transmitted in the client-side deduplication.

sent to the server and the server reads it. This behavior is different than that of the CE algorithm that generates the tag $T \leftarrow H(C)$ on the server. Figure 2 illustrates the upload and download protocols of the HCE1 algorithm. Unlike the CE upload protocol, the client creates a tag that is sent to the server in the HCE1 upload protocol but the server does not return the tag to the client.

The HCE1 scheme can be either a server-side or a client-side protocol, whereas the CE scheme acts as a server-side deduplication protocol. If HCE1 runs as a server-side protocol, the client computes $C \leftarrow E(K, M)$ and sends the server C and t, and the server simply reads t instead of hashing the value of C. If HCE1 is a client-side protocol, the client sends the server t and decides whether to upload C by response from the server. Storing C or updating meta-data depends entirely on the existence of t on the server. The client will upload C only if a tag that is equal to t does not exist at the server. To retrieve and decrypt C, the client keeps t and K.

The HCE1 scheme is vulnerable to both a duplicate-faking attack and an erasure attack because neither of the two parties verifies the integrity of C. If an adversary transmits a forged value C^* instead of a correct value C after sending t, the server will store C^* and not C. When a legitimate client uploads C later, C will be replaced with C^* at the server, and even the client will lose his/her own.

3. Hash and CE with Tag Check (HCE2)

The download protocol of the HCE2 scheme is slightly different than the HCE1 scheme, as shown in Fig. 3(b). After a client receives C and t, the client decrypts $M \leftarrow D(K, C)$ from C, computes $t' \leftarrow G(K) = G(H(M))$, and compares t' with its retained t. If t' is not equal to t, the client rejects C.

Unlike HCE1, the HCE2 scheme prevents an original file from being faked as a poisoned file because this scheme checks the validity of the tag on the client during the decryption routine. However, the HCE2 scheme is still unable to prevent the original file from being erased because the server does not verify the integrity of C.

4. Randomized Convergent Encryption (RCE)

The RCE scheme is very similar to the HCE2 scheme with the exception that RCE uses an additional encryption key. Figure 4 shows a detailed description of the RCE scheme. In the upload protocol, the client derives a tag t from K, selects a random symmetric encryption key L, and encrypts M using L to produce $C1 \leftarrow E(L, M)$. The client also encrypts L using K as a one-time pad, so that computes $C2 \leftarrow K \oplus L$. $C1$ and $C2$ are sent and received as pairs, that is, $C1$ is always accompanied by $C2$ in the upload and download protocol. Regarding the download process, the client computes $L \leftarrow K \oplus C2$ using K and recovers $M \leftarrow D(L, C1)$ using L.

The RCE scheme checks the tag correctness at the client in the download process, similar to HCE2. Accordingly, this scheme provides security protection against only duplicate-faking attacks.

III. Proposed Secure Deduplication

In this section, we construct a new client-side secure

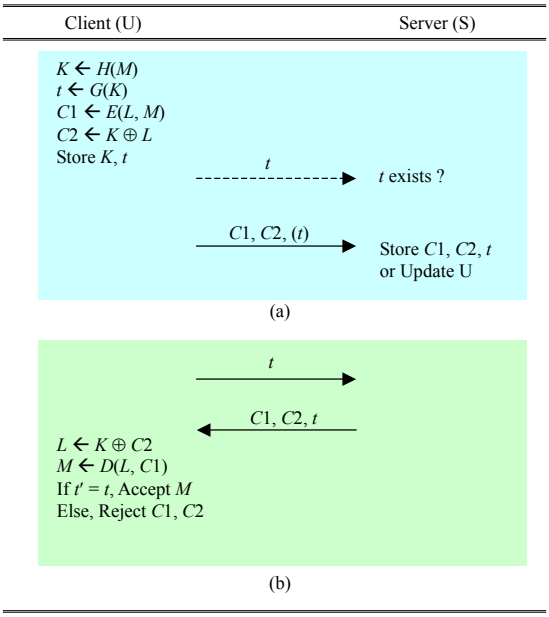


Fig. 4. RCE scheme: (a) upload and (b) download protocols. Dotted arrow is only applied to the client-side deduplication. A parameter in brackets is not transmitted in the client-side deduplication model.

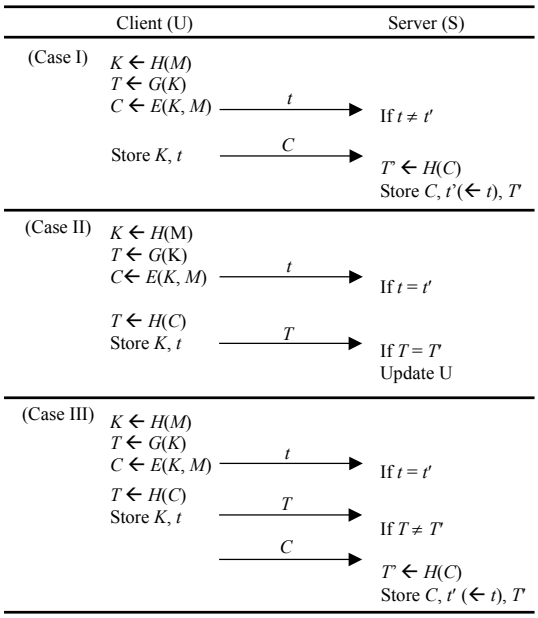


Fig. 5. Upload protocol of the proposed secure deduplication.

deduplication scheme that prevents poison attacks and reduces network traffic.

1. Basic Idea

Our primitive was designed to transfer only non-duplicate files and was thus applied to client-side deduplication. It was also constructed to prevent both duplicate-faking attacks and erasure attacks by creating tags from non-duplicate files at the server. To enhance the security and reduce communication overhead, the proposed primitive uses two types of tags, $t \leftarrow G(K)$, which is derived from a short key K, and $T \leftarrow H(C)$, which is derived from a long ciphertext C.

The client confirms whether a duplicate copy of C exists at the server by sending the first tag t generated with fewer computations than the second tag T. If no tag t' that is equal to t is found at the server, $t \neq t'$ in (Case I) of Fig. 5, the client uploads C. If a tag t' that is identical to t is found, $t = t'$ in (Case II) and (Case III) of Fig. 5, the client transmits the second tag T to the server. The server then searches for a tag T' that is identical to T. If T' exists, $t = t'$ and $T = T'$ in (Case II) of Fig. 5, the server will update the meta-data corresponding to C, and the client will obtain the ownership of C. If T' does not exist, $t = t'$ and $T \neq T'$ in (Case III) of Fig. 5, the client will upload C.

To summarize, the client creates t and T for all files to be uploaded, and the deduplication occurs when both t and T exist at the server. The client finally uploads C only if t or T does not exist at the server. Whenever the server has received a new C, it computes a tag $T' \leftarrow H(C)$ from the non-duplicated C to prevent a poison attack. Therefore, the proposed method can provide the benefits of both security with reasonable network bandwidth. There will be no gain at all to an adversary conducting a poison attack. Because this work focuses on preventing a poison attack during client-side deduplication, other security issues such as a brute-force attack or side channel attack are not considered.

2. Detailed Construction

A. Upload

To achieve secure deduplication, a client (U) and server (S) execute the following protocol, as shown in Fig. 5.

Step 1. U sets $K \leftarrow H(M)$ and $t \leftarrow G(K)$ from M, and produces $C \leftarrow E(K, M)$ under K.

Step 2. U sends t to S.

Step 3. S searches whether t' that is identical to t exists in its data store. If such a t' does not exist, S requests C (go to Step 4). If t' exists, S requests T (go to Step 5).

Step 4. When $t \neq t'$ (Case I),

Step 4-1. U sends C to S and retains K and t. U may encipher K using its own secret key and store it on S.

Step 4-2. S creates $T' \leftarrow H(C)$ from C and stores t', T', and C. The received t is set to t' as $t' \leftarrow t$.

Step 5. When $t = t'$ and $T = T'$ (Case II),

Step 5-1. U computes $T \leftarrow H(C)$ and sends it to S.

Step 5-2. S searches its data store to find out a value T' that is identical to T. If such a T' exists, S simply updates U. Otherwise, S requests C (go to Step 6).

Step 5-3. The existence of both t' and T' means that C is a duplicate. U therefore does not upload C, and retains K and t.

Step 6. When $t = t'$ and $T \neq T'$ (Case III),

Step 6-1. U uploads C, and retains K and t.

Step 6-2. S creates $T' \leftarrow H(C)$ and stores $t' \leftarrow t$, T' and C.

B. Download

U sends t to retrieve C, and S returns C corresponding to t. U can then recover $M \leftarrow D(K, C)$ using K. Our construction does not check the correctness of the tag in the decryption routine.

IV. Analysis

In this section, we analyze our proposal in terms of security, communication costs, and computational complexity. As a result, the proposed primitive (1) provides security that prevents both a duplicate-faking attack and an erasure attack, (2) reduces the amount of data to be transmitted over the network while guaranteeing protection against poison attacks, and (3) requires fewer cryptographic operations to execute the protocol.

1. Security

A poison attack can be mounted by an adversary as follows. Note that the adversary must be the first party to upload the target file because he/she must upload a poisoned file. The adversary may run the protocol using t^* and C^*. As mentioned previously, t^* must be the first tag from the server's viewpoint. As a result of the protocol execution, $\{t^*, T^*, C^*\}$ will be stored at the server. At this time, the tag T^* is computed by the server as $T^* = H(C^*)$. If the same tag t^* is already stored at the server, the adversary should send a second tag T^*. In brief, the adversary can choose a first tag t^* and a poisoned ciphertext C^*, and the corresponding second tag T^* is determined based on the choice of C^* because the server also evaluates T^* from C^*. Suppose that a user wants to store C such that the two tags are t and T. If $t = t^*$, the server will ask the client to send the corresponding second tag. The client then sends T to the server. If $T \neq T^*$, the client will upload his/her file C to the server. Otherwise, if $T = T^*$, the client will obtain the ownership of the stored file C^*.

Recall that the goal of the adversary is to present a state such that the data owners are not aware that they are accessing a poisoned file instead of their original file. As we described previously, the adversary can succeed in deceiving the client only if the following conditions hold:

$$t = t^*, T = T^*, \quad \text{and} \quad C = C^*.$$

The above conditions imply that the adversary can mount a poison attack only if they can find two ciphertexts C and C^* such that $H(C) = H(C^*)$, which contradicts the collision resistance of the underlying hash function $H(\cdot)$. Therefore, the collision resistance of the underlying hash function guarantees the security of the proposed method against a poison attack. In other words, the proposed primitive meets the requirements of the tag consistency problem, which is TC secure and STC secure. It is infeasible to create (M, C) such that $T = H(C) = H(E(K, M)) = H(E(H(M), M))$ but $D(H(M), C)$ is a file that differs from M.

2. Communication

Suppose that a client wants to store n files $\{F_0, F_1, \ldots, F_{n-1}\}$ on a server, and that m $\{F_{i_0}, F_{i_1}, \ldots, F_{i_{m-1}}\}$ of the n files are currently stored on the server, where $0 \le i_0 < i_1 < \cdots < i_{m-1} \le n-1$. Then, the total size of the n files is denoted by N, such that the size of each file F_k is $|F_k|$ and $N = \sum_{k=0}^{n-1} |F_k|$. Similarly, the total size of m duplicate files is computed as $M = \sum_{k=0}^{m-1} |F_{i_k}|$, such that the size of a duplicate file F_{i_k} is $|F_{i_k}|$ and $N \ge M$. Let each length of t, T, and K be $|t|, |T|$, and $|K|$, respectively. We also denote the total amount of traffic transmitted over the network to achieve the secure deduplication, that is, the communication bandwidth cost, as B_{scheme}.

A. CE

The client must upload n files, despite the fact that m files are duplicate files so that they already exist at the server. Consequently, the required traffic for the CE scheme, B_{CE}, is as follows:

$$B_{\text{CE}} = N + n \cdot |T|. \tag{1}$$

B. HCE1, HCE2, and RCE

a. Server-Side Deduplication

The communication cost to store n files for the server-side HCE1 or HCE2 scheme is expressed as (2). The server-side RCE scheme consumes the amount of network traffic shown in (3) due to the use of an additional key.

$$B_{\text{HCE1}_{\text{Server\_side}}} = B_{\text{HCE2}_{\text{Server\_side}}} = N + n \cdot |t|, \tag{2}$$

$$B_{\text{RCE}_{\text{Server\_side}}} = N + n \cdot (|t| + |K|). \tag{3}$$

b. Client-Side Deduplication

The client-side HCE1 and HCE2 protocols require $N - M + (n-m) \cdot |t|$ to upload $n - m$ non-duplicate files and $m \cdot |t|$ to obtain ownership of m duplicate files. Accordingly,

the communication traffic required to store n files is simplified as (4).

$$\begin{aligned} B_{\text{HCE1}_{\text{Client\_side}}} &= B_{\text{HCE2}_{\text{Client\_side}}} \\ &= N - M + (n-m)\cdot|t| + m\cdot|t| \\ &= N - M + n\cdot|t|. \end{aligned} \tag{4}$$

Similarly, the required communication cost for the client-side RCE scheme is as follows:

$$B_{RCE_{\text{Client\_side}}} = N - M + n\cdot|t| + (n-m)\cdot|K|. \tag{5}$$

C. Our Primitive

The proposed primitive requires $N - M + (n-m)\cdot|t|$ to upload $n-m$ non-duplicate files and $m\cdot(|t|+|T|)$ to obtain ownership of m duplicate files. Therefore, the total communication cost becomes (6).

$$\begin{aligned} B_{\text{Ours}} &= N - M + (n-m)\cdot|t| + m\cdot(|t|+|T|) \\ &= N - M + n\cdot|t| + m\cdot|T|. \end{aligned} \tag{6}$$

Note that we exclude the traffic to be transmitted when $t = t'$ and $T \neq T'$ because the probability of the occurrence of this case is negligible if there is no attempt to conduct a poison attack.

Actually, $|t|$, $|T|$, and $|K|$ are negligible compared to N and M. If SHA-256 is used as H and G, $|t|$, $|T|$ and $|K|$ are at most 32 bytes each. For this reason, the required bandwidth for our primitive is nearly identical with the client-side HCE1, HCE2, and RCE schemes.

Figure 6(a) shows that the bandwidth consumption by our primitive decreases linearly as duplicate files increase, whereas the amount of communication traffic required by the CE protocol is constant across all values of the file duplication ratio. In particular, we can see that the proposed primitive applied with a higher rate of duplicate files requires less communication cost as N increases, as shown in Fig. 6(b).

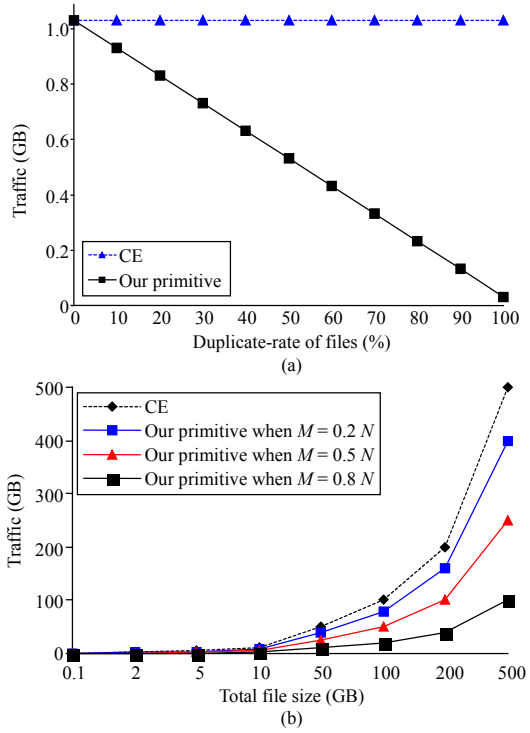


Fig. 6. (a) Network traffic required according to different duplication ratios when n = 1,000 and N = 1 GB, and (b) network traffic required according to the rate of duplicate files as N increases. When providing the same security level, we graphed our primitive and the CE scheme. We did not compare our primitive at a high security level with the client-side HCE1, HCE2, and RCE schemes at a low security level.

3. Computation

The cost to compute $H(\cdot)$ for a long message is much higher than the computational costs of $G(\cdot)$ for a short key even though H and G are the same algorithm. In addition, $E(\cdot)$ and $D(\cdot)$ are also time-consuming processes. Therefore, when the same level of security is provided, a reduction in the number of cryptographic operations of $H(\cdot)$, $E(\cdot)$ and $D(\cdot)$ can provide better performance for both the server and the client.

In the upload protocol, our method conducts $(n-m)\times H(\cdot)$ operations to create tags for only non-duplicate files on the server, whereas the CE scheme requires $n\times H(\cdot)$ operations to process all n files. The overall number of cryptographic operations at the client of our scheme is slightly higher than for the CE scheme with the same security level, and the HCE1, HCE2, and RCE schemes with a lower security level.

In the download protocol, the proposed primitive does not execute any hash operations at the client, whereas the client-side HCE2 and RCE schemes require additional $n\times H(\cdot)$ operations to rule out integrity violation of C, but are still vulnerable to an erasure attack.

We measured the computation time at both the server and the client to compare the performance of the secure deduplication schemes at the same security level. Figure 7 shows the average time consumed for the cryptographic operations on both sides required to upload a file using our primitive and CE scheme. We evaluated the performances on a Windows 7 64 bit system with an Intel Core i7-3770 CPU@3.40 GHz and 8 GB RAM. Our experiments were conducted using SHA256 and AES/ECB/PKCS5Padding based on the Java cryptographic library packaged with jdk1.8.0. It is essential to reduce the computational cost of the server in the upload protocol due to the cross-user deduplication. To store a 4 MB file when $M = 0.5\times N$, our method requires 19.5 ms to execute the hash

Table 1. Evaluation of MLE schemes and the proposed primitive.

	Items to be compared			CE	HCE1	HCE2	RCE	Our primitive
Tag	Used tag			$T \leftarrow H(C)$	$t \leftarrow G(K)$	$t \leftarrow G(K)$	$t \leftarrow G(K)$	$t \leftarrow G(K)$, $T \leftarrow H(C)$
	Party that verifies a tag			Server	N/A	Client	Client	Server
	Protocol that verifies a tag			Upload	N/A	Download	Download	Upload
	Type of deduplication			Server-side	Server-side / client-side	Server-side / client-side	Server-side / client-side	Client-side
Security	Against poison attack	Against duplicate-faking attack (TC secure)		Yes	No	Yes	Yes	Yes
		Against erasure attack (STC secure)		Yes	No	No	No	Yes
Comm. cost	Comm. traffic required to upload	Server-side		$N + n \cdot \lvert T \rvert$	$N + n \cdot \lvert t \rvert$	$N + n \cdot \lvert t \rvert$	$N + n \cdot (\lvert t \rvert + \lvert K \rvert)$	$N - M + n \cdot \lvert t \rvert + m \cdot \lvert T \rvert$
		Client-side			$N - M + n \cdot \lvert t \rvert$	$N - M + n \cdot \lvert t \rvert$	$N - M + n \cdot \lvert t \rvert + (n - m) \cdot \lvert K \rvert$	
Computation	Cryptographic operations	Upload	At server	$n \times H$	N/A	N/A	N/A	$(n - m) \times H$
			At client	$n \times E$	$n \times G + (n - m) \times E$	$n \times G + (n - m) \times E$	$n \times G + (n - m) \times E$	$n \times G + m \times H + n \times E$
		Download	At client	$n \times D$	$n \times D$	$n \times D + n \times H$	$n \times D + n \times H$	$n \times D$

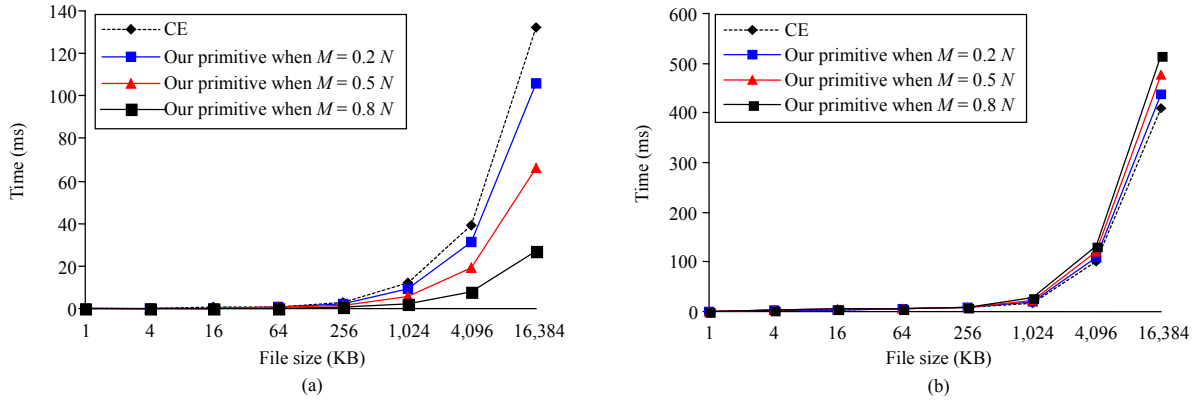


Fig. 7. Average time for cryptographic operations at a (a) server and (b) client required to upload a file, as the file size increases.

computation at the server. However, the CE scheme requires 39 ms. This difference in computation time gradually increases as the number of duplicate files increases. When $M = 0.8 \times N$, our method requires 7.9 ms, whereas the CE scheme still requires 39 ms.

V. Conclusion

We showed that the HCE1, HCE2, and RCE schemes are unable to provide security against poison attacks. We also illustrated that there is a network bandwidth problem with the CE scheme. In Table 1, we provide a summary of the comparison of our proposal relative to the other MLE schemes, from the viewpoint of tag, security, communication, and computational requirements. The results show that our secure deduplication primitive provides security against both duplicate-faking and erasure attacks, requires less network bandwidth, and offers better server performance. The proposed approach will strengthen the advantage of MLE and mitigate its weaknesses.

References

[1] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-Locked Encryption and Secure Deduplication," *Adv. Cryptology - Eurocrypt*, Athens, Greece, May 2013, pp. 296–312.

[2] M. Bellare, S. Keelveedhi, and T. Ristenpart, "DupLESS: Server-Aided Encryption for Deduplicated Storage," *Proc. USENIX Conf. Security*, Washington, DC, USA, Aug. 14–16, 2013, pp. 179–194.

[3] J. Li et al., "Secure Deduplication with Dfficient and Reliable

Convergent Key Management," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 6, June 2014, pp. 1615–1625.

[4] M. Storer et al., "Secure Data Deduplication," *Proc. ACM Int. Workshop Storage Security Survivability*, Alexandria, GA, USA, Oct. 31, 2008, pp. 1–10.

[5] Y. Shin et al., "Efficient and Secure File Deduplication in Cloud Storage," *IEICE Trans. Inform. Syst.*, vol. E97-D, no. 2, 2014, pp. 184–197.

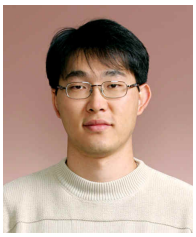
[6] J. Xu, E. Chang, and J. Zhou, "Weak Leakage-Resilient Client-Side Deduplication of Encrypted Data in Cloud Storage," *Proc. ACM SIGSAC Symp. Inform., Comput., Commun. Security*, Hangzhou, China, May 8–10, 2013, pp. 195–206.

[7] N. Kaaniche and M. Laurent, "A Secure Client Side Deduplication Scheme in Cloud Storage Environments," *Proc. Int. Conf. New Technol., Mobility Security*, Dubai, United Arab Emirates, Mar. 30–Apr. 2, 2014, pp. 1–7.

[8] J.R. Douceur et al., "Reclaiming Space from Duplicate Files in a Serverless Distributed File System," *Proc. Int. Conf. Distr. Comput. Syst.*, Vienna, Austria, July 2–5, 2002, pp. 617–624.

[9] D. Harnik, B. Pinkas, and A. Shulman Peleg, "Side Channels in Cloud Services: Deduplication in Cloud Storage," *IEEE Security Privacy*, vol. 8, no. 6, Dec. 2010, pp. 40–47.

[10] S. Halevi et al., "Proofs of Ownership in Remote Storage Systems," *Proc. ACM Conf. Comput. Commun. Security*, Chicago, IL, USA, Oct. 17–22, 2011, pp. 491–500.



Keonwoo Kim received his BS and MS degrees in Electronics Engineering from Kyungpook National University, Daegu, Rep. of Korea, in 1999 and 2001, respectively, and his PhD degree in Computer Science and Engineering from Chungnam National University, Daejeon, Rep. of Korea, in 2013. He has been a principal researcher in ETRI, Daejeon, Rep. of Korea, since 2000. His research interests cover a range of areas in information security, digital forensics, and mobile communications.

Taek-Young Youn received his BS, MS, and PhD degrees from Korea University Seoul, Rep. of Korea in 2003, 2005, and 2009, respectively. He is currently a senior researcher at ETRI, Daejeon, Rep. of Korea. His research interests include cryptography, information security, and data privacy.

Nam-Su Jho received the BS degree in Mathematics from Korea Advanced Institute of Science and Technology, Daejeon, Rep. of Korea, in 1999, and his PhD degree in Mathematics from Seoul National University, Rep. of Korea, in 2007. Since 2007, he has been with the ETRI, Daejeon, Rep. of Korea as a senior researcher. His research interests include cryptography and information theory.

Ku-Young Chang received his BS, MS and PhD degrees in Mathematics from Korea University, Seoul, Rep. of Korea in 1995, 1997, and 2000, respectively. He is currently a principal researcher of Cryptography Research Section at ETRI, Daejeon, Rep. of Korea where his research interests are broadly in the area of applied cryptography and finite field theory.