SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks (2014)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.1159

RESEARCH ARTICLE

Differentially private client-side data deduplication protocol for cloud storage services

Youngjoo Shin[1,2]* and Kwangjo Kim[1]

[1] Department of Computer Science, Korea Advanced Institute of Science and Technology, Yuseong-gu, Daejeon, Korea
[2] The Attached Institute of ETRI, Yuseong-gu, Daejeon, Korea

ABSTRACT

Cloud storage service providers apply data client-side deduplication across multiple users to achieve cost savings of network bandwidth and disk storage. However, deduplication can be used as a side channel by attackers who try to obtain sensitive information of other users' data. We propose a differentially private client-side deduplication protocol. A storage gateway allows efficient data deduplication while reducing the risk of information leakage. Its security can be strongly guaranteed according to the definition of differential privacy. We evaluate the effectiveness and efficiency of the proposed protocol through experiments. Copyright © 2014 John Wiley & Sons, Ltd.

KEYWORDS

cloud storage; secure data deduplication; storage gateway; differential privacy; information leakage-resilient protocol

***Correspondence**

Youngjoo Shin, The attached institute of ETRI, Yuseong-gu, Daejeon, Korea.
E-mail: s.youngjoo@kaist.ac.kr

1. INTRODUCTION

Most cloud storage service providers utilize data deduplication to save the cost of network bandwidth and disk storage [1]. Client-side (or source-based) data deduplication technique eliminates duplicate copies of data (or files) across multiple users before uploading them. It has been known that this technique is not secure, because by monitoring and analyzing network traffic, adversaries can use (client-side) deduplication as a side channel to obtain sensitive information of other users' data [2,3]. Recently, some solutions have been proposed to prevent such information leakage, but several problems still remain unresolved.

Harnik *et al.* scheme [2] and its security-enhanced version [4] obfuscate the occurrence of deduplication by randomizing the event with certain probability. This randomized approach, however, causes huge network bandwidth consumption owing to the unnecessary file upload. In addition, the schemes are based on the strong assumption that all individual files are independent of each other. This leads to our new attack, denoted by *related-files attack*, that we propose in this paper. Security of these approaches [2,4] are weakened by *related-files attack* in which an adversary can take advantage of knowledge of correlations among files. Olivier *et al.* proposed another solution for secure deduplication [5]. Their idea is to run the deduplication protocol at a home router provided by an Internet service provider (ISP) and mix network traffic for cloud storage with other service traffic. This solution, however, lacks flexibility because the fact that it requires an ISP home router limits its uses to a specific service model.

Allowing efficient client-side deduplication while reducing the risk of information leakage is still an open problem. In this paper, we address this problem and propose a secure client-side deduplication protocol. Our solution utilizes a storage gateway, which is a network appliance that resides at the customer's premises and provides APIs to access the remote cloud storage server. Because they simplify interactions with cloud storage services, storage gateways are gaining popularity and being deployed as a key component in various cloud service models, such as public, private, or hybrid cloud computing [6]. The key idea of the proposed solution is that a storage gateway handles user requests for file uploads and performs data deduplication on them on behalf of the users (or clients). In this way, the network traffic for each user request is merged together at the storage gateway's Wide Area Network (WAN) interface. Without generating unnecessary network traffic, this approach weakens the link between the deduplication event and the amount of actually transferred. This idea is similar to of Olivier *et al.* solution [5], but using storage gateways gives flexibility for adapting to various cloud service models.

For robust security, we apply a differentially private mechanism to our proposed protocol. Differential privacy is a security notion that has been presented in database security literature to design privacy-enhanced statistical databases [7–9]. By exploiting differential privacy, the proposed protocol strongly guarantees that the presence or absence of individual files in the cloud storage is hard to infer from any side channel attacks, including *related-files attack*.

The contributions of this paper are as follows. First, we discuss the security weakness of the previous schemes [2,4] that are based on the randomized approach. Second, we propose a storage gateway-based solution that guarantees robust security, while providing network efficiency and flexibility. Third, we analyze its security under the definition of differential privacy and evaluate its effectiveness by our experiments.

The rest of this paper is organized as follows: In Section 2, we review the previously proposed schemes and describe their security weakness. In Section 3, details of our storage gateway-based deduplication solution are described. In Section 4, we present a security analysis and the experiment results of the proposed protocol. Finally, in Section 5, we give the summary of this paper.

2. RELATED-FILES ATTACK

2.1. *Related-files attack* on Harnik *et al.* scheme

We briefly review Harnik *et al.* scheme [2] and present its security weakness.

2.1.1. The protocol description.

For every file F, a cloud storage server assigns a threshold t_F chosen randomly in a range $[2, d]$, where d is a security parameter that might be known public. The cloud server keeps a counter c_F, which represents the number of previous uploads of a copy of F. When a client uploads a new copy of F, the server performs (client-side) deduplication if at least t_F copies of F have been previously uploaded (i.e., $c_F \geq t_F$) or if the copy is uploaded by an identical user who has previously uploaded F; no deduplication event occurs otherwise. This protocol is illustrated in Figure 1(a).

In order to analyze its security, let us consider the following three types of events where the adversary is about to identify whether a copy of F was uploaded.

(i) ($2 < t_F < d$) The adversary finds out that a deduplication event occurs after uploading $2 < t < d$ copies of F. This could be due to two cases:

  (i-1) F already exists, and the adversary uploads $t = t_F - 1$ copies of F and
  (i-2) F does not exist and the adversary uploads $t = t_F$ copies of F.

(ii) ($t_F = 2$) The adversary finds out that a deduplication event occurs after uploading a single copy of F. It is the case that $t_F = 2$ and a single copy of F has been previously uploaded by another user.

(iii) ($t_F = d$) The adversary finds out that a deduplication event occurs after uploading d copies of F. It is the case that no copy of F was previously uploaded and $t_F = d$.

For F with $2 < t_F < d$, it can be shown that the events of two cases (i-1) and (i-2) occur with equal probabilities. Hence, a deduplication event gives an adversary no information of the existence of F in the storage. On the other hand, for F with $t_F = 2$ or $t_F = d$, the adversary can easily learn the existence of F by uploading just one or d copies of the file. Note that the two events (ii) and (iii) are mutually exclusive and each event happens with probability $\frac{1}{d-1}$. As a result, with probability $1 - \frac{1}{d-1}$, the scheme described previously leaks no information, which enables an adversary to distinguish between the case that a single copy of F was previously uploaded and the case that the file was not uploaded.

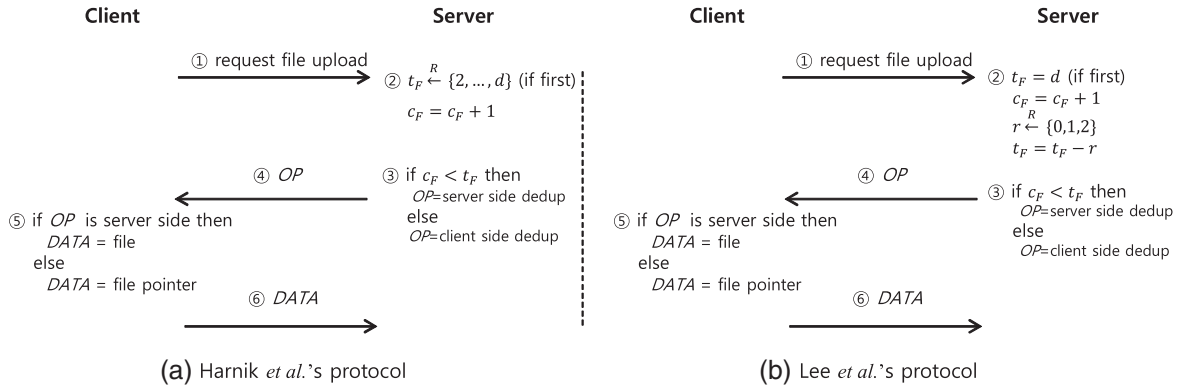

Figure 1. Randomized client-side deduplication protocol.

DOI: 10.1002/sec

2.1.2. *Related-files attack*.

The security of Harnik *et al.* scheme stands on an implicit assumption that all files are stored independently. This assumption is too strong, because in real environments, some files are likely to be correlated to each other and thus stored at the same server together. For instance, some executable files may coexist to construct a software package. We also consider that some document files with the same content in different formats, such as doc, pdf, and xml, coexist to support various document viewers.

Related-files attack exploits this correlations among files. By uploading not only F but also other files related to F, the adversary can infer the existence of F with high probability. For Harnik *et al.* scheme, the probability that the threshold of at least one is 2 or d among n-related files is $p = 1 - \left(1 - \frac{1}{d-1}\right)^n$, and the probability increases as n increases. Figure 2 shows the relationship between the number of related files (n) and the probability (p) that the adversary successfully obtains information of F via *related-files attack* varying the security parameter d. For example, with 10 files related to F (i.e., $n = 10$), the adversary can determine the existence of F with probability $p = 0.45$ (assume that $d = 18$), which is much higher than $p = 0.06$ with no related files (i.e., $n = 1$).

2.2. *Related-files attack* on Lee *et al.* scheme

Lee *et al.* [4] observed that the probability of Harnik *et al.* scheme that information of the existence of a file is leaked to the adversary is not negligible in security parameter d. Following the observation, they proposed a security-enhanced version of Harnik *et al.* solution.

2.2.1. The protocol description.

In Lee *et al.* scheme, a threshold t_F is initially set to d for every file F, where d is a security parameter. For each upload of a copy of F, the server randomly chooses $r \in \{0, 1, 2\}$ and calculates $t_F = t_F - r$ instantly. Then, similar to Harnik *et al.* scheme, deduplication occurs at client side if $c_F \geq t_F$ or the copy of F is uploaded by the identical user; otherwise, deduplication is performed at sever-side. Figure 1(b) illustrates this protocol.

As analyzed in [4], the adversary's success probability can be minimized when the security parameter d is selected as a multiple of 3. The probability that the adversary learns information of the existence of F is $\left(\frac{1}{3}\right)^{d/3} + \left(\frac{1}{3}\right)^d$.

2.2.2. *Related-files attack*.

The security of Lee *et al.* scheme [4], however, stands on the same assumption as Harnik *et al.* method that files are not related to each other and stored independently. The probability that the adversary finds out deduplication events for at least one out of n related files is $p = 1 - \left\{1 - \left(\frac{1}{3}\right)^{d/3} - \left(\frac{1}{3}\right)^d\right\}^n$, which increases as the number of related files increases. Figure 3 shows the relationship between the number of related files (n) and the probability (p) varying the security parameter (d). As shown through the graph in Figure 3, Lee *et al.* solution is still vulnerable to *related-files attack*.

3. THE PROPOSED PROTOCOL

3.1. Background

The notion of differential privacy was introduced by Dwork *et al.* [7–9]. It is a standard privacy notion for

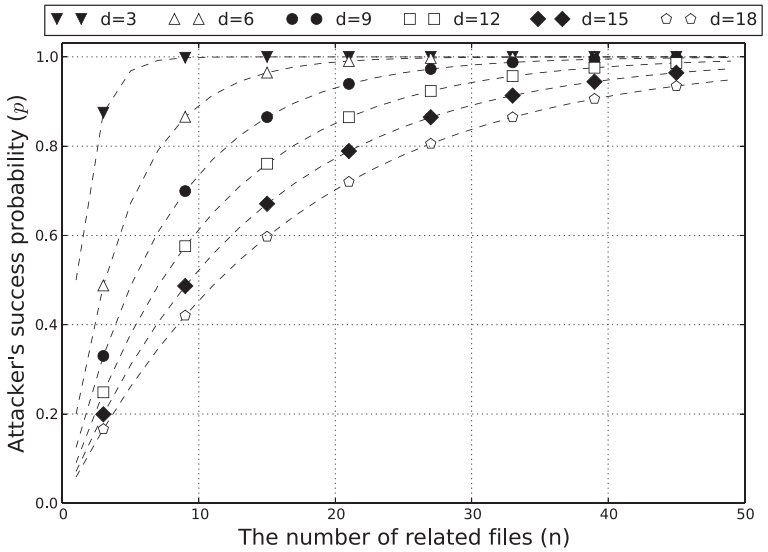

Figure 2. Success probability of *related-files attack* against Harnik *el al.* scheme.

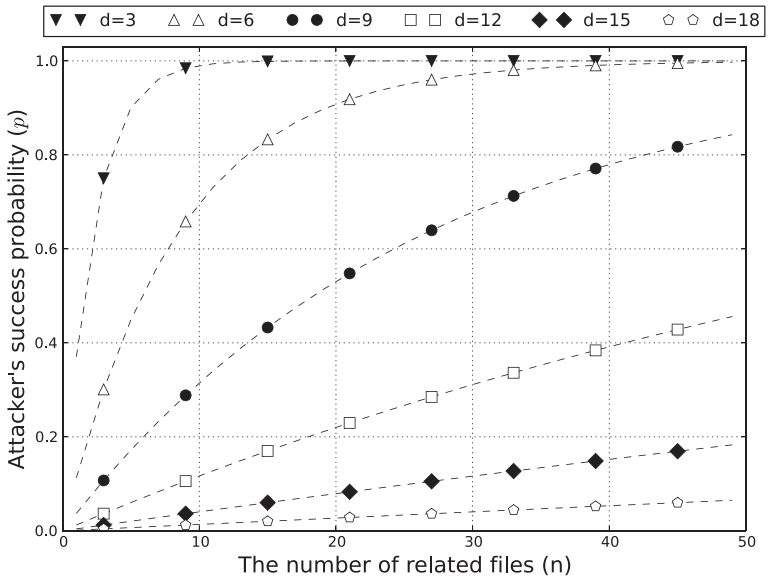

Figure 3. Success probability of *related-files attack* against Lee *el al.* scheme.

release functions over sets of data items (or databases). Differential privacy is defined for randomized functions (i.e., distributions over the set of possible outputs) and can be formally stated as follows.

Definition 1. *A randomized function $Q : D \to R$ satisfies ϵ-differential privacy if for all $r \in R$ and for all data sets D and D' that differ in a single item*

$$Pr\{Q(D) = r\} \leq e^{\epsilon} \cdot Pr\{Q(D') = r\}$$

The definition states that the probability of any result of the randomized function Q is almost independent of whether any individual item among the data set is present in the input. For each item, it is almost as if the item was not used in the computation of the function, a very strong baseline for privacy.

Differential privacy is actually preserved by adding noise to the output of the function. Intuitively, the noise introduces uncertainty about the real value of the output, which naturally makes the real values of the inputs also uncertain to an adversary. The basic method for implementing differential privacy mechanism is to use Laplacian noise, which is a symmetric exponential distribution [7]. The mechanism is additive, that is, given a function, it approximates its output by computing the function exactly and then adding noise sampled from a specific distribution.

Comparing with alternative privacy definitions, differential privacy gives strong security guarantee. Unlike the notion of differential privacy, many other privacy notions do not provide a direct guarantee or are even vulnerable to auxiliary information that an adversary might know. For example, the notion of k-anonymity [10], which provides security on releasing data such that the identity of individual data items remains private, is weaker than the notion of differential privacy. That is, k-anonymity is provided if the information for each item cannot be distinguished from at least k-1 other items. However, this definition gives no guarantee if the adversary has knowledge of some auxiliary information about the data sets [11].

3.2. System and attack model

We consider a general cloud storage service model that involves the following three entities (Figure 4).

- Cloud storage server (CSS): this is owned by a cloud service provider and provides many storage resources as a service to its users through WAN (or the Internet).
- Storage gateway (GW): this is a local disk-attached network server that resides at the customer's on-premises site and provides an interface to access data at CSS. Every user request for a file upload or download is processed by GW, which in turn performs data deduplication and transfers the actual user data to CSS.
- Users (U): a user outsources its data to CSS through GW. Data deduplication becomes transparent to U because the process is performed on GW.

An adversary $\mathcal{A}$, acting as a user by creating its own account or compromising the accounts of others , interacts with GW by uploading or downloading some data. During the interactions, $\mathcal{A}$ could view all network traffic and its content between $\mathcal{A}$ (including compromised users) and GW. We assume that a connection from GW's WAN interface to CSS is encrypted by Secure Socket Layer (SSL) or Transport Layer Security (TLS) protocol. As recent results [12,13] show that SSL/TLS keeps the content of communications securely from being sniffed by adversaries unless

DOI: 10.1002/sec

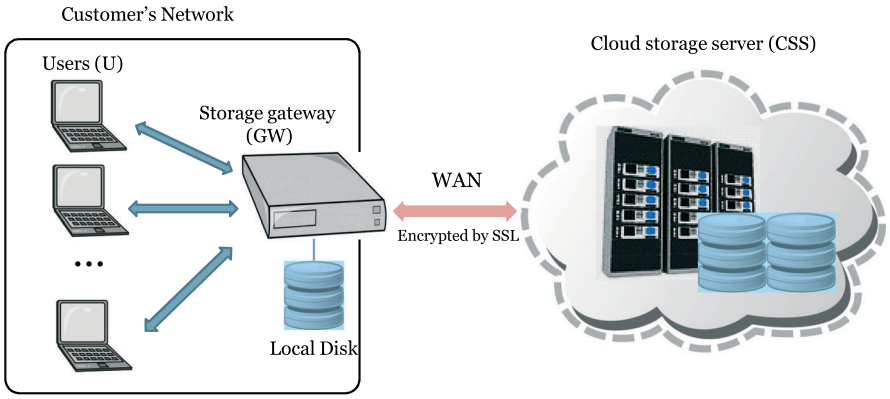

Figure 4. The system model of the proposed protocol.

some flaws in its implementation exist, $\mathcal{A}$ can monitor only the amount of traffic at the link between GW and CSS, but not its content. The goals of $\mathcal{A}$ are (1) to determine the existence of a file of interest in CSS and eventually (2) to learn the content of the file. $\mathcal{A}$ may have auxiliary information that several files coexist with the file of interest. Monitoring the network traffic, $\mathcal{A}$, will mount a sophisticated traffic analysis attack with powerful computing resources to achieve its goals.

3.3. Design goal

According to the attack model described previously, $\mathcal{A}$ views a deduplication as an oracle and will use it when mounting the attack. That is, $\mathcal{A}$ continues file uploading and observing the amount of network traffic that eventually occurs during the attack. Thus, the deduplication can be modeled as a function $Q_S(F)$, which takes a file F as input and outputs the amount of network traffic. The output will be determined by whether a storage S contains F or not. Let us denote a storage without F by S_1 and a storage with F by S_2 (i.e., $S_2 = S_1 \bigcup \{F\}$). The attack can be stated that $\mathcal{A}$ is trying to find out whether the storage S is S_1 or S_2 by querying a function $Q_S(F)$ one or more times. The primary design goal of the proposed protocol is to make $Q_S(F)$ randomized and satisfy ϵ-differential privacy so that $\mathcal{A}$ hardly distinguishes between S_1 and S_2.

Definition 2. *We say that storage S is a set of files in CSS. A randomized function $Q_S(F)$ gives ϵ-differential privacy if for all storages S_1 and S_2 that differ in a single file F (i.e., $S_2 = S_1 \bigcup \{F\}$) and all $0 \leq \tau \leq |F|$,*

$$Pr\left\{Q_{S_2}(F) = \tau\right\} \leq e^{\epsilon} \cdot Pr\left\{Q_{S_1}(F) = \tau\right\}$$

The proposed protocol is expected to achieve the following security and performance goals.

- *Single file privacy (ϵ-differential privacy)*: the proposed protocol should give ϵ-differential privacy so that $\mathcal{A}$ cannot distinguish between two storages S_1 and S_2, where $S_2 = S_1 \bigcup \{F\}$. That is, the proposed protocol should prevent $\mathcal{A}$ from using deduplication as a side channel.
- *Related files privacy*: the proposed protocol should prevent $\mathcal{A}$ from obtaining any information even when knowing some related files $\{F_1, F_2, \ldots, F_n\}$ for any $n > 1$. That is, $\mathcal{A}$ should not gain a higher advantage in distinguishing S_1 from $S_2 = S_1 \bigcup \{F_1, F_2, \ldots, F_n\}$ than distinguishing it from $S_2 = S_1 \bigcup \{F\}$.
- *Efficiency*: network traffic overhead that is caused by the proposed protocol should be minimized. We also require that the proposed protocol should not significantly degrade the system performance on GW.

3.4. Protocol description

The proposed deduplication protocol is implemented on GW. The main idea is that (1) for hiding the deduplication from $\mathcal{A}$, GW runs a differentially private algorithm when deciding the volume of data to be sent to CSS; (2) for minimizing the network overhead, GW maintains a queue, which we denote T, in which data accepted from U is temporarily stored before transmission, and if needed, GW uses data from T as noise traffic instead of generating dummy traffic. The proposed protocol is composed of two operations: *File Upload* and *File Download*.

3.4.1. File upload operation.

U uploads a file F to GW. During this upload, deduplication is not run, and all bytes of F are always sent to GW. Once F is accepted, GW runs Algorithm 1 to perform deduplication and transfer the data to CSS. Let us denote by $\{b_1, b_2, \ldots, b_{|F|}\}$ a sequence of bytes of F and by $\{c_1, c_2, \ldots, c_{|T|}\}$ a sequence of bytes of data stored in T. The details of Algorithm 1 are as follows.

(1) Put $F = \{b_1, b_2, \ldots, b_{|F|}\}$ into the local disk of GW.
(2) Communicate with CSS to check if F has already been stored in CSS. In detail, CHECKFILEEXISTS() algorithm calculates $h(F)$, a hash value of F,

Algorithm 1

```
1: procedure UPLOAD(F)
2:     Buffer ← ∅
3:     FileExists ← CHECKFILEEXISTS(F)
4:     τ ← GETTRANSFERSIZE(|F|, FileExists)
5:     if FileExists is false then
6:         Buffer ← {b_1, b_2, ..., b_τ}
7:         ENQUEUE(T, {b_{τ+1}, b_{τ+2}, ..., b_{|F|}})   ▷ Put |F| – τ bytes of F into T
8:     else
9:         {c_1, c_2, ..., c_t} ← DEQUEUE(T, τ)   ▷ Get τ bytes of data from T
10:        Buffer ← {c_1, c_2, ..., c_t}
11:        if t < τ then
12:            DummyBytes ←R {0,1}^((|τ|–|t|)×8)
13:            Buffer ← Buffer || DummyBytes
14:        end if
15:    end if
16:    TRANSFER(Buffer)   ▷ Send data in Buffer to CSS
17: end procedure
```

and sends $h(F)$ to CSS. Then, it receives an answer $FileExists \in \{true, false\}$ from CSS.

(3) Compute $\tau (0 \leq \tau \leq |F|)$, the amount of data to be actually transferred to CSS. In order to determine τ, GETTRANSFERSIZE() (Algorithm 2) uses $Poi(\lambda)$, which generates a *Poisson*-distributed random number from a finite interval $[0, 1]$ with a mean λ.

Algorithm 2

```
1: procedure GETTRANSFERSIZE(|F|, FileExists)
2:     if FileExists is true then
3:         α ←R Poi((1+ε)/2)   ▷ ε is a privacy parameter (0 ≤ ε ≤ 1)
4:     else
5:         α ←R Poi((1–ε)/2)
6:     end if
7:     τ ← ⌈α|F|⌉   ▷ |F| is the size of F in bytes and 0 ≤ α ≤ 1
8:     return τ
9: end procedure
```

(4) If F does not exist in CSS (i.e., a deduplication event does not occur), all bytes of F should be transferred to CSS. A subset of bytes $\{b_1, b_2, \ldots, b_\tau\}$ of F is moved to *Buffer* from the local storage for transmission. The remaining bytes $\{b_{\tau+1}, b_{\tau+2}, \ldots, b_{|F|}\}$ are then enqueued into T by running ENQUEUE() algorithm.

(5) If F already exists in CSS (i.e., a deduplication event occurs), it is not necessary to transfer F to CSS. Hence, τ bytes of data $\{c_1, c_2, \ldots, c_t\}$ are dequeued from T by running DEQUEUE() and then put onto *Buffer* for transmission. If there are not enough data in T (i.e., $t < \tau$), dummy bytes are randomly generated and padded to *Buffer* so that the total size becomes equal to τ.

(6) Finally, the data in *Buffer* are sent to CSS by running TRANSFER().

3.4.2. File download operation.

U sends GW a request for downloading F. Upon receiving, GW retrieves F in the storage of CSS. If found, GW returns the requested file F to U. In order to enhance performance for file downloading, GW may first find F in the local storage. If F is found on GW, it can be immediately sent to U.

Security consideration. Adversaries may exploit the file download operation when trying to learn the existence of F on CSS. If GW keeps files at the local storage for some time (i.e., for average upload time taken for each file on GW to complete its upload to CSS) regardless of the deduplication result, adversaries will have no information on F.

4. EVALUATION

4.1. Security analysis

We analyze the security of the proposed protocol in terms of two requested security properties. For analysis, the proposed protocol is modeled as a randomized function $Q_S(F)$ in this section. Suppose that an adversary algorithm $\mathcal{A}$ tries to learn information on the file of interest in CSS.

4.1.1. Single file privacy.

$\mathcal{A}$ has no information on the occurrence of deduplication by monitoring the link between $\mathcal{A}$ and GW because all bytes of the file are always sent to GW at the link. At WAN link of GW, the content of the network is not visible to $\mathcal{A}$ because of encryption; thus, $\mathcal{A}$ has no option but to conduct traffic analysis by querying $Q_S(F)$. The output of $Q_S(F)$, which we denoted by τ in the previous section, is actually determined by Algorithm 2. By the definition of *Poisson* distribution, $Pr\{Poi(\lambda) = x\} = \lambda^x e^{-x}/x!$. It can be shown that the proposed protocol satisfies the security that is described in *Definition 2* and hence gives single file privacy.

$$\frac{\Pr\{Q_{S_2}(F) = \tau\}}{\Pr\{Q_{S_1}(F) = \tau\}} = \frac{\Pr\left\{Poi\left(\frac{1-\epsilon}{2}\right) = \alpha\right\}}{\Pr\left\{Poi\left(\frac{1+\epsilon}{2}\right) = \alpha\right\}} = \frac{\left(\frac{1-\epsilon}{2}\right)^\alpha}{\left(\frac{1+\epsilon}{2}\right)^\alpha} \frac{e^{\frac{-1+\epsilon}{2}}}{e^{\frac{-1-\epsilon}{2}}} = \left(\frac{1-\epsilon}{1+\epsilon}\right)^\alpha e^\epsilon \leq e^\epsilon$$

DOI: 10.1002/sec

Single file privacy means that for any two storages that differ on a single file, the proposed protocol will release approximately the same volume of network transmission on both storages. This property guarantees that the presence or absence of an individual file will not affect the output of the proposed protocol significantly. Thus, it prevents the adversary $\mathcal{A}$ from using deduplication as a side channel.

4.1.2. Related files privacy.

Let us denote three storages: $S_1, S_2 = S_1 \bigcup \{F\}$, and $S_3 = S_1 \bigcup \{F_1, F_2, \ldots, F_n\}$, where $F \in \{F_1, F_2, \ldots, F_n\}$. It is obvious that the computation of $Q_S(F)$ is independent of the rest of the files and $\Pr\left\{Q_{S_2}(F) = \tau\right\} = \Pr\left\{Q_{S_3}(F) = \tau\right\}$. The ratio of probabilities is therefore

$$\frac{\Pr\left\{Q_{S_2}(F) = \tau\right\}}{\Pr\left\{Q_{S_1}(F) = \tau\right\}} = \frac{\Pr\left\{Q_{S_3}(F) = \tau\right\}}{\Pr\left\{Q_{S_1}(F) = \tau\right\}}$$

This result implies that the statistical difference of $Q_S(F)$ between S_1 and S_3 is always equal to the difference between S_1 and S_2. Hence, the advantage that $\mathcal{A}$ succeeds in distinguishing between S_1 and S_3 (i.e., with knowledge of related files $\{F_1, F_2, \ldots, F_n\}$) is not greater than the probability of distinguishing between storages S_1 and S_2, which differ in a single file.

4.2. Experiments

We conducted experiments for evaluating the effectiveness and efficiency of the proposed protocol. Our test bed consisted of two servers, which acted as GW and CSS, with an Intel Core processor i5 running at 2.8 GHz and 4 GB RAM memory with a 2 TB hard disk. One server acting as the GW is located at local area network and connected with a PC, which simulates multiple clients. The other server is located remotely over the Internet and acting as the CSS. The data set used in our experiments consists of files including Windows system files, office documents, and media files, totaling up to 3 TBs. The data set's duplicate ratio (i.e., a proportion of redundant files in the data set) is 27.5%. Figure 5 shows cumulative file size distribution of the data set. We split the data set into two groups of equal size and put one group into CSS and used the other group for uploading at client side during the experiment. Uploading behavior by users was simulated such that the upload event followed exponential distribution with its frequency varying from 10 to 600 uploads per 10 min.

4.2.1. Network efficiency.

We first evaluated the network efficiency by measuring traffic overhead and comparing the measurements with the previous randomized schemes of Harnik *et al.* and Lee *el al.* For the experiment, we simulated the previous schemes with all network traffic passing through GW. In the proposed and the previous schemes, traffic overhead is measured at the outgoing (WAN) interface of GW to CSS. Figure 6 shows the amount of accumulated traffic overhead over time with varying security parameters. "Traffic overhead" of the graph refers to the amount of extra bandwidth consumption used to obfuscate the occurrence of deduplication. In the proposed protocol, dummy traffic is generated and transferred to CSS only if there are not enough data in T. Hence, the extra bandwidth consumption is quite small as compared to that of the previous schemes, in which copies of duplicate data are always transferred until it reaches the threshold. With the greatest security guarantee ($\epsilon = 0$), the proposed protocol incurs 48 GBs in

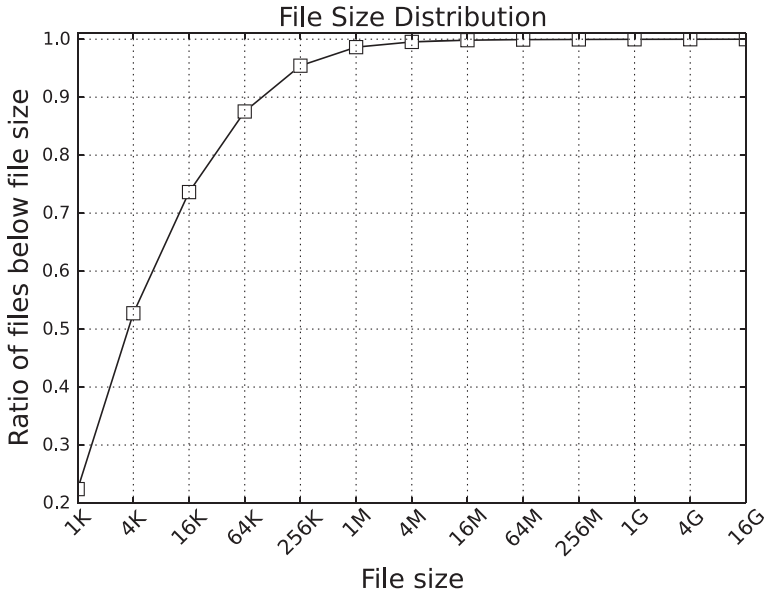

Figure 5. Cumulative file size distribution in the data set.

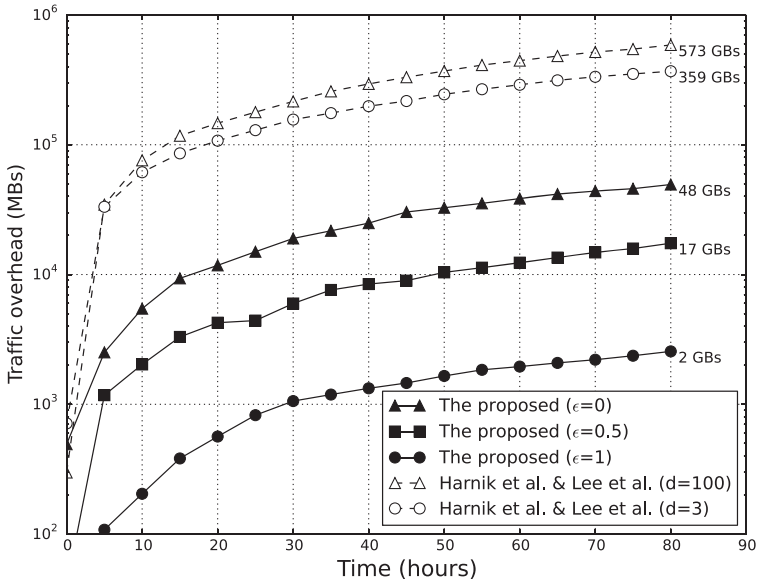

Figure 6. Network transmission overhead over time.

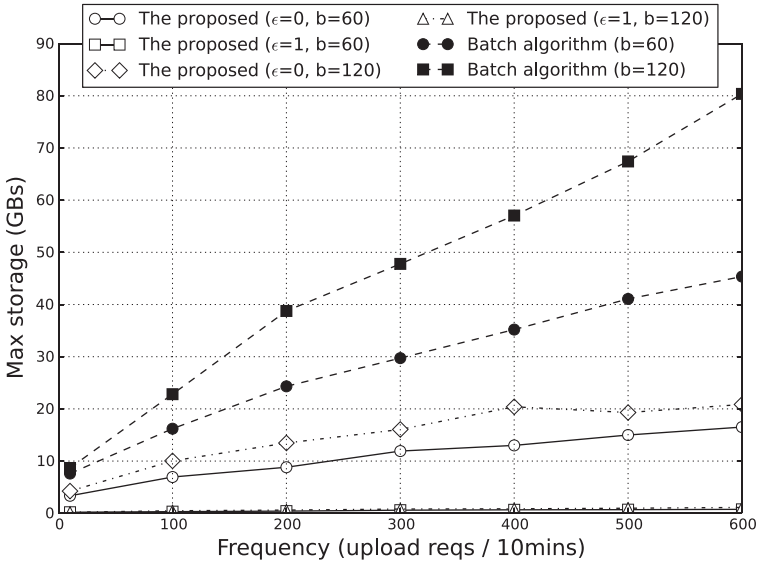

Figure 7. Storage overhead (*b* refers to batch interval in minutes).

total, which is about 13% of the traffic overhead incurred by the previous schemes with the lowest security guarantee ($d = 3$).

4.2.2. System performance.

We also evaluated the system performance on GW by measuring two metrics: required local disk capacity and average processing time for each request. Both metrics are sufficient to measure the required system resources on GW for running the proposed protocol. For comparison, we considered a batch algorithm that is similar to the proposed protocol but performs in a naive way. The batch algorithm immediately accepts user files and performs deduplication but transfers them to CSS periodically every given batch interval. For ease of comparison, we ran the proposed protocol such that at every batch interval, all the remaining data in T were uploaded to CSS. Figures 7 and 8 show two measurements under different batch intervals (60, 120 min) and varying upload frequencies. In Figure 7, "Max storage" refers to the greatest amount of data stored in the local disk on GW during the experiment, which indicates the disk capacity required to run these schemes. In Figure 8, "Avg. processing time" refers to average time taken for each upload request to complete its upload to CSS. The

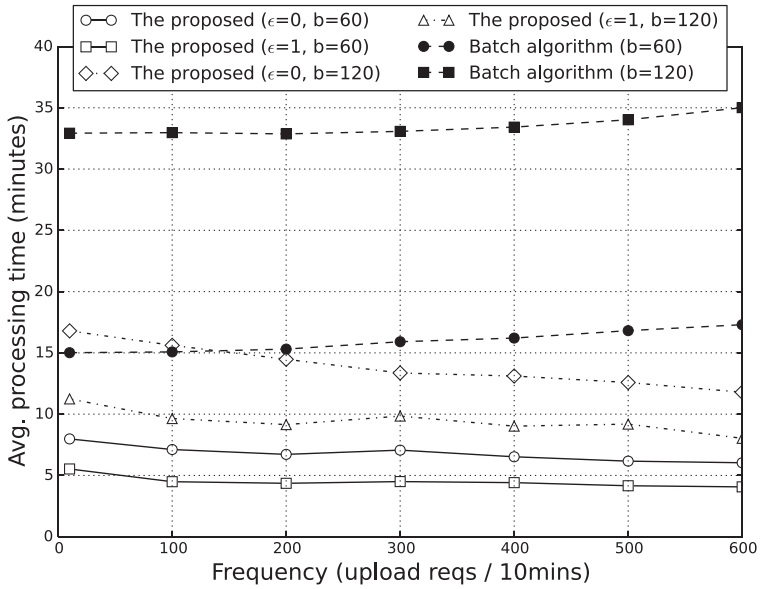

Figure 8. Average processing time.

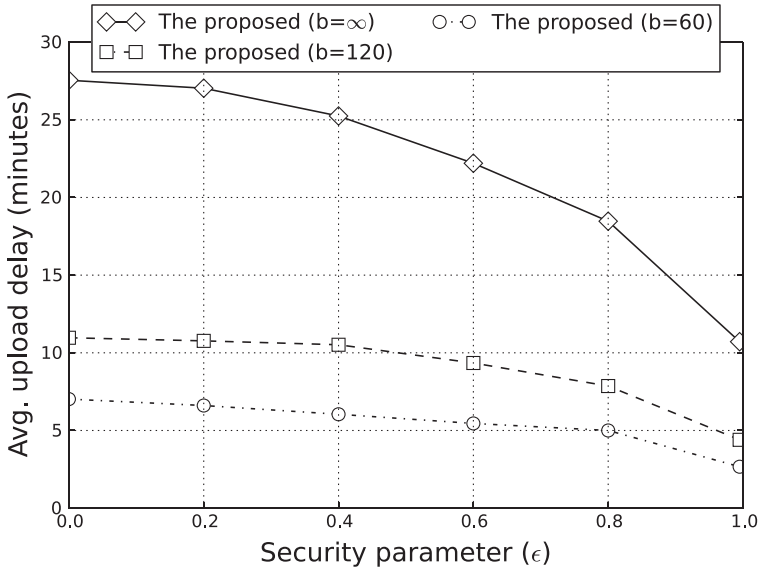

Figure 9. Tradeoff between security parameter(ϵ) and average upload delay time (upload frequency is 600 per 10 mins).

results of this experiment indicate that the proposed protocol uses a decreasing amount of system resources on GW as the number of users (i.e., upload frequency) increases as compared to the batch algorithm.

While the proposed protocol can achieve higher security, some delay may be incurred during file upload operation, because some parts of data will be delayed on GW. In order to investigate this tradeoff between the security and the performance, we measured average upload delay time (i.e., average time taken for each remaining part of data on GW to be uploaded to CSS) varying security parameters. Figure 9 shows a result of the experiment. With no batch interval ($b = \infty$) and greatest security guarantee ($\epsilon = 0$), average upload delay is about 27 mins. The delay can be greatly reduced by increasing ϵ or applying batch intervals to the proposed protocol.

5. CONCLUSION

In order to achieve cost savings of network bandwidth and disk storage, cloud storage service providers apply client-side (or source-based) data deduplication techniques. Unfortunately, deduplication can be used as a side chan-

nel by adversaries who try to obtain sensitive information of other users' data. Several solutions have been proposed to prevent such information leakage. However, these solutions are based on a strong assumption that all individual files (or data) are independent of each other.

Observing this assumption, we proposed new attack, which is called *related-files attack*, to the previous approaches. In order to mitigate such an attack, we proposed a storage GW-based secure client-side deduplication protocol. A storage GW is a network appliance that provides access to the remote cloud server and simplifies interactions with cloud storage services, and is used in various cloud service delivery models such as public, private, and hybrid cloud computing. The proposed solution, by utilizing the storage GW as an important component in the system design, achieves greater network efficiency and architectural flexibility while also reducing the risk of information leakage.

For more robust security guarantee, we applied a differential private mechanism to the proposed protocol. Differential privacy is a security notion that has been used for designing privacy-enhanced databases. By exploiting a differential private mechanism, the proposed protocol strongly guarantees that the presence or absence of individual files in the cloud storage is hard to infer from any side channel attacks including *related-files attack* that has been proposed in the paper.

For validation of the effectiveness and efficiency of the proposed protocol, we conducted several experiments using real data sets. The network efficiency and the system performance of the proposed protocol were evaluated by measuring traffic overhead, required local disk capacity and average processing time for each request. The experiments showed that the proposed protocol outperforms the previous approaches.

By exploiting a differential private mechanism, the security against any side channel attacks is strongly guaranteed.

Acknowledgements

This research was funded by the MSIP, Korea in the ICT R&D Program 2014 [1391104001] and the KUSTAR-KAIST Institute, KAIST, Korea.

REFERENCES

1. Russell D. Data deduplication will be even bigger in 2010. *Gartner* 2010, Available at: http://www.gartner.com/doc/1297513 [accessed on 24 March 2014].
2. Harnik D, Pinkas B, Shulman-Peleg A. Side channels in cloud services: deduplication in cloud storage. *IEEE Security and Privacy Magazine* 2010; **8**: 40–47.
3. Mulazzani M, Schrittwieser S, Leithner M, Huber M, Weippl E. Dark clouds on the horizon: using cloud storage as attack vector and online slack space, *Proceedings of USENIX Security Symposium (SEC'11)*, San Francisco, CA, 2011; 65–76.
4. Lee S, Choi D. Privacy-preserving cross-user source-based data deduplication in cloud storage, *Proceedings of International Conference on ICT Convergence*, Jeju, Korea, 2012; 329–330.
5. Olivier, Neumann C, Montalvo L, Defrance S. Improving the resistance to side-channel attacks on cloud storage services, *Proceedings of International Conference on New Technologies, Mobility and Security (NTMS'12)*, Istanbul, Turkey, 2012; 1–5.
6. Zaffos S, Couture A. Hybrid cloud gateway appliances expand cloud storage use cases. *Gartner* 2011, Available at: http://www.gartner.com/doc/1516718 [accessed on 24 March 2014].
7. Dwork C, McSherry F, Nissim K, Smith A. Calibrating noise to sensitivity in private data analysis, *Proceedings of Conference on Theory of Cryptography (TCC'06),* LNCS, vol. 3876, New York, NY, 2006; 265–284.
8. Dwork C. Differential privacy: a survey of results, *Theory and Applications of Models of Computation,* LNCS, vol. 4978, Xi'an, China, 2008; 1–19.
9. Dwork C, Smith A. Differential privacy for statistics: what we know and what we want to learn. *Journal of Privacy and Confidentiality* 2010; **1**(2): 135–154.
10. Sweeney L. k-anonymity: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 2002; **10**(5): 557–570.
11. Narayanan A, Shmatikov V. Robust de-anonymization of large sparse datasets, *Proceedings of IEEE Symposium on Security and Privacy (SP'08)*, Oakland, CA, 2008; 111–125.
12. Fahl S, Harbach M, Muders T, Baumgärtner L, Freisleben B, Smith M. Why eve and mallory love android: an analysis of android ssl (in)security, *Proceedings of ACM Conference on Computer and Communications Security (CCS'12)*, Raleigh, NC, 2012; 50–61.
13. Krawczyk H, Paterson KG, Wee H. On the security of the tls protocol: a systematic analysis, *Advances in Cryptology—CRYPTO'13,* LNCS, vol. 8042, Santa Barbara, CA, 2013; 429–448.