An Enhanced Remote Data Checking Scheme for Dynamic Updates

Lin Dong[1], Jinwoo Park[1], Junbeom Hur[2] and Ho-Hyun Park[1*]
[1] School of Electrical and Electronics Engineering, Chung-Ang University
84, Heukseok-ro, Dongjak-gu, Seoul 156-756, Korea
[e-mail: hohyun@cau.ac.kr]
[2] School of Computer Science and Engineering, Chung-Ang University
84, Heukseok-ro, Dongjak-gu, Seoul 156-756, Korea
[e-mail: jbhur@cau.ac.kr]
*Corresponding author: Ho-Hyun Park

Received January 21, 2014; revised April 10, 2014; accepted May 3, 2014; published May 29, 2014

Abstract

A client stores data in the cloud and uses remote data checking (RDC) schemes to check the integrity of the data. The client can detect the corruption of the data using RDC schemes. Recently, robust RDC schemes have integrated forward error-correcting codes (FECs) to ensure the integrity of data while enabling dynamic update operations. Thus, minor data corruption can be recovered by FECs, whereas major data corruption can be detected by spot-checking techniques. However, this requires high communication overhead for dynamic update, because a small update may require the client to download an entire file. The Variable Length Constraint Group (VLCG) scheme overcomes this disadvantage by downloading the RS-encoded parity data for update instead of the entire file. Despite this, it needs to download all the parity data for any minor update. In this paper, we propose an improved RDC scheme in which the communication overhead can be reduced by downloading only a part of the parity data for update while simultaneously ensuring the integrity of the data. Efficiency and security analysis show that the proposed scheme enhances efficiency without any security degradation.

Keywords: Cloud computing, remote data checking, DPDP, R-DPDP, VLCG

http://dx.doi.org/10.3837/tiis.2014.05.014

1. Introduction

The rising popularity of cloud computing and storage-outsourcing has led clients to outsource their data and programs to untrusted servers. Thus, clients lose direct control of data and programs. Clients cannot prevent some attacks and accidents involving their data. Therefore, in the cloud computing environment, it is necessary for clients to check the integrity of the data stored in servers [1].

Remote data checking (RDC) schemes [2] allow auditors to check the integrity of data stored at a third-party server using spot checking. A general RDC scheme contains two phases, store and audit. In the store phase, the client generates metadata and a modified file and stores the modified file in the server. In the audit phase, the auditor generates challenges and sends the challenges to the server. The server calculates the proofs of the challenges using the modified file and sends these proofs to the auditor. Then, the auditor verifies the received proofs using the metadata. In general RDC schemes, the client plays the role of an auditor. RDC schemes include Provable Data Possession (PDP) [3], Proofs of Retrievability (PoR) [4], [5], High-Availability and Integrity Layer (HAIL) [6], and Dynamic Provable Data Possession (DPDP) [7], [8], [9], [10].

PDP, PoR, and HAIL are suitable only for static files [11], whereas DPDP can perform dynamic updates (including insert, modify, and delete). However, both of these RDC schemes can protect data only against major corruption of data based on spot checking. Thus, they cannot provide protection against arbitrary small amounts of data corruption.

A robust auditing scheme [12] is introduced which mitigates arbitrary amounts of data corruption. Robustness is usually realized by integrating forward error-correcting codes (FECs) [13] with remote data checking mechanisms [12], [14], [15]. Thus, with minor data corruption, there is no damage to the data because the corruption can be recovered by FECs; on the other hand, with major data corruption, the client can easily detect corruption using spot-checking techniques [12].

Robust Dynamic Provable Data Possession (R-DPDP) [16] is a remote auditing scheme that adds robustness to DPDP. Two R-DPDP constructions have been proposed, the πR-D and Variable Length Constraint Group (VLCG) schemes. πR-D [16] is an early enhancement of the πR scheme [12]. πR-D not only provides robustness, but also supports dynamic updates including insertions, modifications, and deletions. However, it requires high communication overhead for insertions / deletions (a small update may demand the download of an entire file). VLCG [16] decreases the communication overhead mainly in two ways: i) assigning symbols to constraint groups based on the values of the symbols instead of the indices of the symbols; ii) reducing insertion / deletion to append / modify when updating the RS-encoded parity data. Therefore, VLCG needs to download only the RS-encoded parity data instead of the entire file for update. However, to ensure the integrity of the data, VLCG must also download all of the parity data including the useless data even for small updates.

In this paper, we propose an improved RDC scheme to reduce the communication overhead for update. In our scheme, instead of downloading all of the RS-encoded parity data [17] for any small update, the clients download only a part of the RS-encoded parity data. The downloaded parity data includes the data that needs to be updated and some redundant data to avoid finding the association between symbols and constraint groups. To ensure δ-robustness [12] (to be defined in Section 2), it is necessary to hide the association between the symbols and constraint groups. Thus, the clients need to download not only the

parity symbols that are in the affected constraint groups, but also some redundant portions of the parity data.

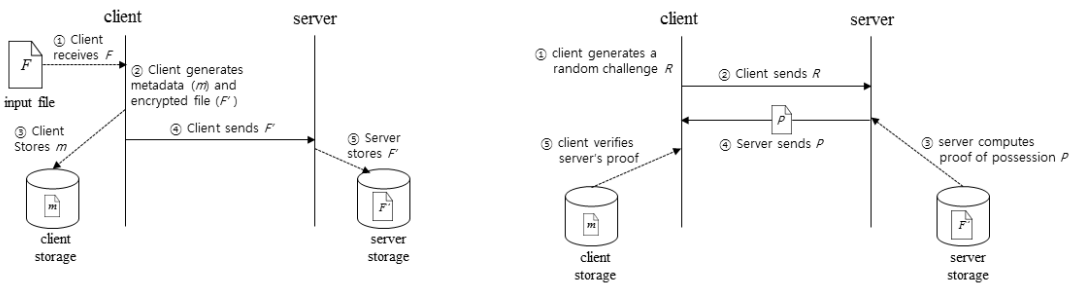
The main contributions of this work can be summarized as follows:

1. The proposed scheme reduces communication overhead by downloading only a part of the RS-encoded parity data for dynamic update.
2. The proposed scheme calculates the minimum download ratio of the total parity data in order to guarantee δ-robustness.

The experimental results show that our scheme requires less communication overhead when the update size is small relative to the file size. Further, they show that with varying file sizes, our scheme can ensure δ-robustness by calculating and downloading the minimum amount of data so that security is not degraded.

2. Background and Related Work

There has been substantial research on remote data checking. PDP [3] has provided a probabilistic proof that the server can store the client's data without retrieving it. In PDP, before transmitting a file to the server, the client generates a piece of metadata and stores it locally. Then, the client transmits the file to the server and deletes it locally. The client may encrypt the file prior to the transmission to the server. Before deleting the file locally, the client should ensure that the server has stored the file successfully. The client can achieve this purpose through a challenge-response protocol for the data. The challenge indicates specific blocks for which the client wants to prove its possession. When the client finds it is necessary to check the integrity of the data stored in the server, the client sends a challenge to the server. Instead of accessing all the blocks of the file, the client generates a random challenge by sampling the data blocks stored in the server. The server generates the corresponding proof based on the data and transmits it to the client. Thus, the client can verify this proof using the metadata. This process is shown in Fig. 1. However, the PDP scheme has an important drawback – high resource overhead. The client must store numerous metadata that is linear in the number of challenges.



(a) Pre-processing and storing files **(b)** Process of verifying server possession

Fig. 1. Protocol for provable data possession

Proof of Retrievability (PoR) [4], [5] is an auditing scheme similar to the PDP. In PoR, instead of storing metadata, the client stores only a key that is used to encode the file. After encrypting the file, the client generates a set of sentinel values and embeds these values into the encrypted file. The server stores only the received file without knowing the positions of the embedded sentinel values. The client challenges the server by specifying the positions of a collection of sentinel values. If the server has corrupted a large fraction of the file, the

server will be unable to generate a complete proof for the challenge with high probability. Consequently, the client knows that the original file has been corrupted. PDP and PoR are applicable only to static files.

Until now, neither the PDP nor the PoR has been able to allow public and private verifiability concurrently since different setup procedures and metadata sets are required. Hanser and Slamanig [18] have proposed a simultaneous privately and publicly verifiable PDP protocol to support both private and public verifiability simultaneously. This new construction is based on elliptic curves, with the same setup procedure and metadata set being used for private and public verifiability. This scheme is more efficient in terms of storage and communication overhead, as well as computational effort for the client and the server.

Erway et al. [7] have proposed the DPDP scheme, which supports full dynamic operations (append, insert, delete, and modify). In DPDP, dynamic operations are realized based on rank-based authenticated dictionaries built over a skip list and RSA tree. Although the computational complexity increases because of the need to support dynamic updates, DPDP is efficient practically in terms of communication and computational overhead for update.

All of these schemes can protect data only against a large amount of data corruption. R-DPDP is a scheme obtained by adding the robustness property to the DPDP scheme. In R-DPDP, corrupted data can be recovered even if the corruption is minor. Because the R-DPDP scheme is closely related to our scheme, we will explain it in the subsequent sections.

2.1 Forward Error-Correcting Codes

Forward error correction [13] is a technique used to improve data reliability in the fields of data communication, information theory, digital signal processing, and the like. Error correction can be performed by introducing redundant data, called error-correcting code. Redundancy allows the receiver to detect a limited number of errors that may occur anywhere in the message, and often to correct these errors without retransmission. Reed-Solomon (RS) code [17] is a variation of FECs. The notation (n, k) RS code denotes an RS code that encodes a message of k symbols into a codeword of n symbols including $d = n - k$ redundant symbols; thus, up to d erasures can be corrected.

2.2 Robust Dynamic Provable Data Possession

R-DPDP [12], [16] applies (n, k) RS code over the entire file to achieve robustness. It divides the file F into k-symbol chunks, and (n, k) RS code is applied to every chunk. Therefore, every chunk is expanded into n-symbol codewords in which the first k symbols are the original data symbols and the following $d = n - k$ symbols are the parity symbols corresponding to the original data symbols. These n-symbol codewords containing k original data symbols and their corresponding $n - k$ parity symbols are defined as a constraint group [12], [16] (a detailed explanation of constraint groups will be given in Example 2). RS encoding and decoding are based on Cauchy matrices [16], [19]. In Cauchy RS updating, modify / append operations have lower bandwidth overhead than insert / delete operations. In [12], [16], a δ-robust auditing scheme is defined as follows:

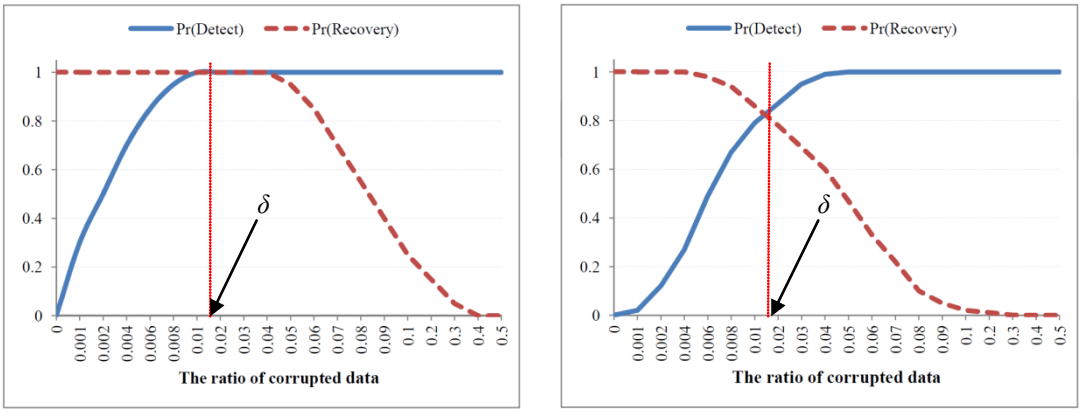
Definition 1. A robust auditing scheme RA is a tuple (C, T), where C is a remote data checking scheme for a file F, and T is a transformation that yields $\tilde{F}$ when applied on F. We say that RA provides δ-robustness when

— the auditor will detect with high probability if the server corrupts more than a δ-fraction of $\tilde{F}$ (protection against corruption of a large portion of $\tilde{F}$);

— the auditor will recover the data in F with high probability if the server corrupts at most a δ-fraction of $\tilde{F}$ (protection against corruption of a small portion of $\tilde{F}$).

Generally, the high probability in Definition 1 is almost one. Similar to the previous PDP, POR, DPDP, and R-DPDP schemes, the client acts as an auditor. The following example helps to understand the definition of δ-robust auditing scheme more clearly.

Example 1: In **Fig. 2**, **(a)** and **(b)** show the probability of detection and recovery with varying sizes of corrupted data in two different auditing schemes. In both of these schemes, for simple representation, we set δ = 0.015. In the first scheme, as shown in **(a)**, the probability of detecting the corruption is one, while the ratio of corrupted data is more than δ, whereas in the condition that the ratio of corrupted data is not more than δ, the probability of recovery is one. Thus, this scheme can protect data against varying sizes of corruption; hence, the first scheme is a δ-robust auditing scheme. In this scheme, there is an overlapped interval in the horizontal axis in which the probability of detection and recovery are both one; any value of the horizontal axis in the overlapped interval can be δ. However, in the second scheme, as shown in **(b)**, if the attacker corrupts δ-fraction of the data, the probability of detecting this corruption is about 0.85, and the probability of recovery from the corruption is around 0.8. According to this result, the second scheme is not a δ-robust auditing scheme. Furthermore, the second scheme cannot ensure δ-robustness irrespective of how we set the value of δ on the horizontal axis.



(a) A scheme that can ensure δ-robustness **(b)** A scheme that cannot ensure δ-robustness

Fig. 2. Two schemes differing with regard to δ-robustness

In a robust auditing scheme, error detection and recovery are performed by spot checking and RS coding, respectively. In RS coding, symbols are encoded per constraint group. Therefore, if the relationship between symbols and a constraint group is revealed, an attacker can corrupt the constraint group intentionally. If only a few constraint groups are damaged,

the damage may not be recovered; furthermore, it may not be detected by the auditing mechanism. Then, this attack succeeds and δ-robustness is not guaranteed. Chen and Curtmola proposed two robust DPDP schemes, πR-D and VLCG [16], both of which guarantee δ-robustness.

2.2.1 πR-D

The πR-D scheme [16] adds robustness on the basis of the DPDP scheme. The DPDP scheme [7] consists of four phases: Setup, Challenge, Update, and Retrieve. In the Setup phase, parity data P is generated by applying the RS code to file F. The augmented file $D = F||P$ is encrypted and then pre-processed using the DPDP algorithms [7]. All other phases except the Update phase use the DPDP algorithms directly.

The Update phase can progress with three different operations: insert, delete, and modify. In the insert / delete operations, the data symbols to be inserted / deleted are assigned to constraint groups. The contents of a constraint group are decided according to the indices provided by a pseudo-random permutation (PRP) ψ [20]. Since the operations of inserting or deleting a data symbol will affect the indices of the subsequent data symbols in the entire file, the client needs to download the entire file F and recalculate the parity P according to a new set of constraint groups. The data symbols to be inserted / deleted and the new parity data is returned to the server by the client. The update bandwidth factor is $\alpha=\frac{|F|}{|D|}$, which approaches one in practice [16].

In the modify operations, a client downloads the data blocks to be modified and all of the parity data. The client decrypts P, restores the original order using ψ^{-1}, and then updates the parity data symbols that belong to the same constraint group with the modified data symbols. The update bandwidth factor is $\alpha=\frac{|P|}{|D|}$. From the performance point of view, the update bandwidth factor of the modify operations is much lower than that of the insert and delete operations.

2.2.2 Variable Length Constraint Group

In the πR-D scheme, the insert / delete operations need to download the entire file F, because PRP ψ is applied to the indices of data symbols. In order to overcome this drawback, Chen and Curtmola proposed the VLCG scheme [16]. VLCG relies on two additional main insights. First, a cryptographic hash function $h_K(b)$ [21] is applied to the value of symbol b to decide the index of the constraint group to which symbol b belongs. Thus, insert / delete operations cannot affect the constraint groups of other symbols. The cryptographic hash function is usually assumed to be collision resistant [21]. Therefore, we assume that k consecutive data symbols are assigned to different groups by the hash function, respectively. Second, to minimize the bandwidth overhead, VLCG simplifies an insert operation to an append operation, and a delete operation to a modify operation, when updating the RS-coded parity data.

In an insert operation, VLCG updates parity symbols in the constraint group to which the insert symbol is assigned as if the symbol were appended to the end of the data symbols in the constraint group. A delete operation is more complex. The parity symbols in the constraint group to which the delete symbol is assigned are updated as if the symbol were

modified to have the value zero. A modify operation is the most complex, because if a symbol is modified to a different new value, it may be assigned to a different constraint group. The old symbol must be deleted from its constraint group, and the new symbol must be inserted in a new constraint group. Appending a data symbol to the Cauchy RS code or modifying a data symbol of the Cauchy RS code needs to download only the parity symbols in the updated constraint group. However, to ensure δ-robustness, the client must download all of the parity data P. Thus, the bandwidth factors of all the update operations are $\alpha = \frac{|P|}{|D|}$.

Example 2: For ease of presentation, we illustrate Cauchy RS encoding and decoding using a (6, 4) RS code as an example (i.e., n = 6, k = 4, d = 2). There are four constraint groups, and each block contains two symbols; thus, there are four parity blocks. **Fig. 3** shows an example where 16 data symbols in the original file are assigned to four constraint groups ($CG1$, $CG2$, $CG3$, $CG4$) using a hash function $h_K(b_i)$. Now, the client generates the $d = 2$ ($= n - k$) parity symbols of each constraint group using the Cauchy RS encoding. Consequently, each constraint group contains four original data symbols and two parity symbols. For example, in constraint group $CG1$, the first four symbols 2, 7, 9, 15 are the original data symbols and the following $P1$ and $P2$ are the two parity symbols of this constraint group. The eight parity symbols of the four constraint groups are appended to the end of the original file F after performing PRP ψ. Then, suppose a client wants to update (insert, delete, and modify) a symbol b_i (= 5) in the original file F. The client finds that this symbol is located in constraint group four ($CG4$) using the hash function $h_K(5) = 4$; hence, the parity symbols $P7$ and $P8$ must be updated. However, irrespective of which parity blocks the parity symbols $P7$ and $P8$ belong to, all the parity data must be downloaded.

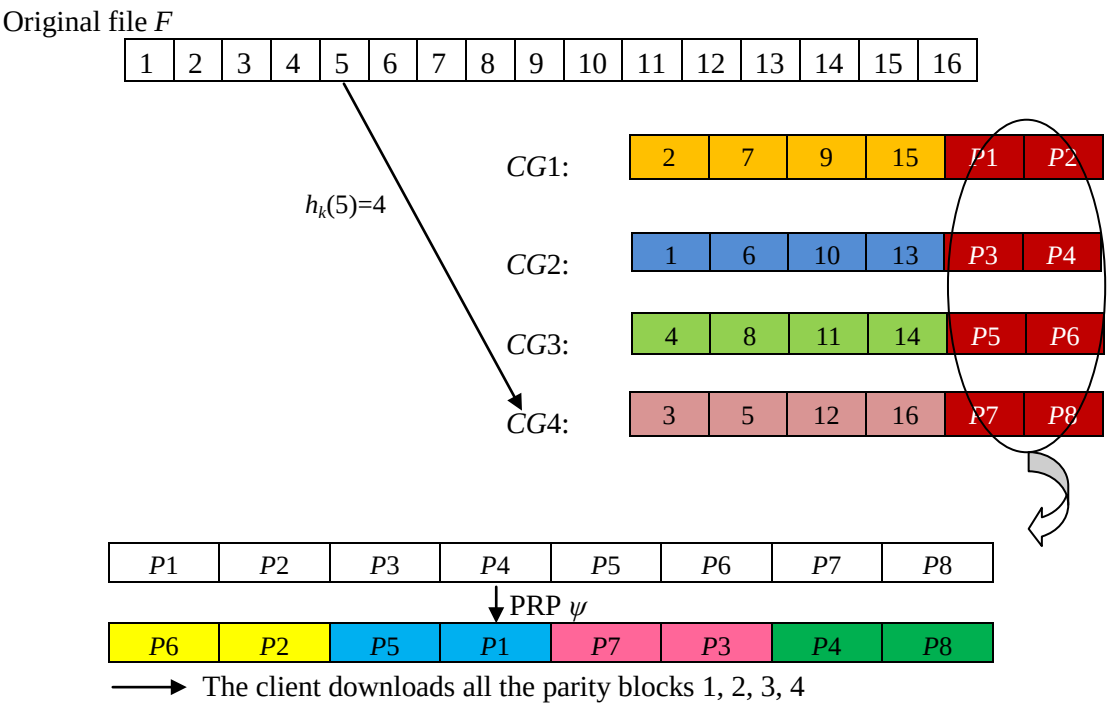


Fig. 3. The method of downloading all parity blocks in VLCG

As shown above, the VLCG scheme, which is the newest R-DPDP scheme, still needs to download all of the parity data for any small update. There has not yet been any further effort to reduce the communication overhead for dynamic update. Therefore, in this paper, we propose an enhanced remote data checking scheme that reduces the communication overhead for updates. This scheme is based on the R-DPDP scheme.

3. Downloading Part of the Parity Blocks

3.1 Basic Scheme

In the πR-D and VLCG schemes, clients must download all of the parity data in the Update phase to ensure δ-robustness. If the number of updated symbols is small, the probability of finding the association between symbols and constraint groups becomes very low. On the contrary, if there are many updated symbols, the probability increases. (The detailed explanation will be given in Section 3.2.) In either case, δ-robustness is guaranteed [12], [14].

However, we argue that when the number of updated symbols is very small, downloading all the parity data is very inefficient with regard to communication overhead. We define the threshold of the probability that an attack succeeds as σ (in [12], with σ set to a value less than 10^{-10}). If the number of updated symbols is small, the probability that an attack succeeds can be reduced to less than σ by downloading only part of the parity data for the updated symbols. Thus, the δ-robustness in Definition 1 can be redefined with σ as follows:

Definition 2. A robust auditing scheme RA is a tuple (C, T), where C is a remote data checking scheme for a file F, and T is a transformation that yields $\tilde{F}$ when applied on F. We say that RA provides δ-robustness when

— the auditor will detect with high probability (**the probability of not detecting the damage is less than σ**) if the server corrupts more than a δ-fraction of $\tilde{F}$ (protection against corruption of a large portion of $\tilde{F}$);

— the auditor will recover the data in F with high probability (**the probability of not recovering the damage is less than σ**) if the server corrupts at most a δ-fraction of $\tilde{F}$ (protection against corruption of a small portion of $\tilde{F}$).

In this section, we present an efficient remote data-checking scheme that downloads parity symbols in proportion to the number of updated symbols. Our first approach is to download only the parity symbols of the constraint groups to which the updated symbols belong. However, when the number of updated symbols is small, the association between a constraint group and the symbols of the group can be easily revealed. If an attacker deletes all the updated and parity symbols of a constraint group, the deletion may not be recovered and detected. Therefore, we need some redundancy in the downloaded parity symbols.

The second approach is to download not only the parity symbols of the constraint group but also all the other symbols in the blocks to which the parity symbols belong. Then, revealing the association between a constraint group and the symbols of the group can be mitigated. In order to understand better the difference between our scheme and VLCG, the following example is given.

Example 3: In this example, all the conditions are the same as in Example 2. The client needs to update the parity symbols of $P7$ and $P8$. As shown in **Fig. 4**, instead of downloading all the parity symbols, the client finds the locations of the parity symbols of $P7$ and $P8$ after permutation. Then, the client downloads the blocks in which these parity symbols are located, namely, the third and fourth blocks.

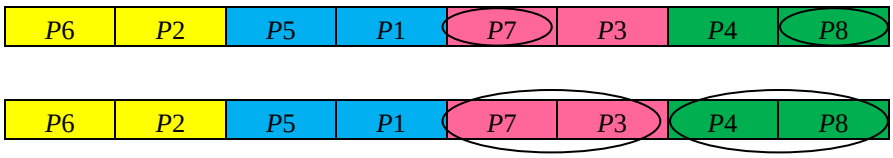


Fig. 4. The method of downloading part of parity blocks

When the number of updated symbols is small, the probability of a successful attack may still be higher than σ despite the client downloading the parity blocks instead of parity symbols. In order to solve this problem, we may require additional download for redundancy. Therefore, we suggest the third approach. In the third approach, i) we set the minimum number of parity blocks to be downloaded to ensure δ-robustness according to the number of updated symbols, and ii) if the number of parity blocks obtained from the second approach is less than the minimum number, we download additional parity blocks that are randomly selected using a pseudo-random function. The next section treats a method to calculate the minimum number of downloaded parity blocks.

The following are some notations to be used in the next section:

$\boldsymbol{p}$ is the total number of parity symbols.

$\boldsymbol{f}$ is the number of data symbols, i.e., original file symbols.

$\boldsymbol{n}$ is the number of symbols in a constraint group.

$\boldsymbol{k}$ is the number of data symbols in a constraint group.

$\boldsymbol{d}$ is the number of parity symbols in a constraint group.

$\boldsymbol{g}$ is the number of updated constraint groups.

$\boldsymbol{X_R}$ is the number of corrupted parity symbols.

$\boldsymbol{C_R}$ is the number of symbols in parity data checked by spot checking.

$\boldsymbol{\alpha}$ is the update bandwidth factor defined as

α = the amount of data downloaded for updating one file block / the total number of data at the server.

$\boldsymbol{Damage_R}$ is the ratio of corrupted parity symbols among the total parity symbols.

$Damage_R$ = the number of corrupted parity symbols / the total number of parity symbols

$\boldsymbol{Damage_{min}}$ is the minimum ratio of corrupted parity symbols that can be detected.

$\boldsymbol{Download_R}$ is the ratio of downloaded parity symbols among the total parity symbols.

$Download_R$ = the number of downloaded parity symbols / the total number of parity symbols

$\boldsymbol{Download_{min}}$ is the minimum download ratio of parity symbols to ensure δ-robustness.

3.2 Minimum Download Ratio

We consider security only in the Update phase because all the other phases are the same as in VLCG. During update operations, the attacker can discover the updated location of the file F easily and know that all the parity symbols of the updated constraint groups are contained in the downloaded parity blocks. Therefore, if the attacker corrupts the updated symbols in the file F and the parity symbols in the locations of the downloaded parity blocks, the probability of corrupting more than d symbols in one updated constraint group will increase. When more than d symbols in one updated constraint group are corrupted, this constraint group is damaged and may not be recovered [12]. Further, if this damage is not detected, the attack succeeds.

We mentioned that downloading part of the parity data P affects the probability of discovering the association between the updated parity symbols and the updated constraint groups. However, we can determine the minimum download ratio of parity symbols such that the probability of a successful attack is less than σ.

To determine the minimum download ratio of parity symbols, we will first calculate the probability of detecting an attack and then the probability of recovering from the attack. The client detects server misbehavior in the data blocks and parity blocks. However, according to the collision resistance property of cryptographic hash functions, we can assure that there is no more than one updated data symbol assigned to the same constraint group. Thus, even if the attacker corrupts all the updated data symbols, there is only one data symbol to be corrupted in any one constraint group. Then, we must see whether the d parity symbols in this constraint group are corrupted. The d parity symbols of the constraint group play an important role in the security of the constraint group in the Update phase, as will be demonstrated later. Thus, we calculate only the probability of detecting the attack in the parity data $Pr(detect_R)$.

By the definition of σ in Section 3.1, we have

$$1 - Pr(detect_R) \le \sigma . \tag{1}$$

The client detects server misbehavior in the parity data after a challenge in which it requests for proof for C_R parity symbols. According to [12], the detection probability is represented by the following:

$$Pr(detect_R) \ge 1 - \left(1 - \frac{X_R}{p}\right)^{C_R} . \tag{2}$$

In formula (2), $p(= d \cdot f / k)$ and X_R represent the number of all parity symbols and the number of corrupted parity symbols in a file, respectively. C_R denotes the number of symbols spot-checked in the parity data .

Since $Damage_R = X_R / p$, formula (2) can be rewritten as follows:

$$Pr(detect_R) \ge 1 - (1 - Damage_R)^{C_R} \;\; => \;\; 1 - Pr(detect_R) \le (1 - Damage_R)^{C_R} . \tag{3}$$

To guarantee δ-robustness, formula (1) must be satisfied. If we ensure that

$$(1 - Damage_R)^{C_R} \le \sigma , \tag{4}$$

then formula (1) can be established. Formula (4) can be rewritten as follows:

$$Damage_R \ge 1 - \sigma^{\frac{1}{C_R}} . \tag{5}$$

Through formula (5), we can determine the minimum ratio of corrupted parity symbols that can be detected as follows:

$$Damage_{min} = 1 - \sigma^{\frac{1}{C_R}}. \tag{6}$$

If the ratio of corrupted parity data is more than $Damage_{min}$, this corruption will be detected. However, we cannot simply regard $Damage_{min}$ as the minimum download ratio $Download_{min}$, because it may not be able to ensure δ-robustness. For example, in the condition of $Download_{min} = Damage_{min}$, when the attacker corrupts part of the downloaded parity symbols that are less than $Damage_{min}$, this corruption will not be detected, but the probability of causing damage to the data is high. To illustrate this case, the following example is given.

Example 4: Suppose the size of the original file is 128000 * 4 KB, encoded using a (140, 128) RS code, and the update size is one symbol. If the number of symbols spot-checked is 1188 * 4 KB as in [12], then $C_R = \frac{12}{140} \cdot 1188 \cdot 4 \cdot 1024 = 417090$. We can calculate $Damage_{min} = 5.5 \times 10^{-5}$ using formula (6). When the client downloads $Damage_{min} \cdot p = 5.5 \times 10^{-5} \times 49152000 = 2703$ parity symbols, and an attacker corrupts half of the downloaded parity symbols, the probability of damaging all 12 parity symbols in the constraint group is $\binom{Damage_{min} \cdot p \cdot \frac{1}{2}}{d} \Big/ \binom{Damage_{min} \cdot p}{d} = \binom{2703 \times \frac{1}{2}}{12} \Big/ \binom{2703}{12} \approx \frac{1}{4096} \approx 2.44 \times 10^{-4}$. In this condition, the probability of detecting this damage is very low, because the ratio of corrupted parity data among the total parity data is less than $Damage_{min}$. In addition, the probability of not recovering this constraint group is 2.44×10^{-4}, which does not meet the condition of less than σ (= 10^{-10} [7]) in Definition 2. Thus, this attack can succeed. Therefore, the minimum download ratio of parity blocks must be greater than $Damage_{min}$.

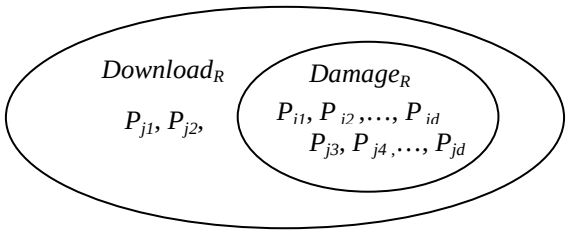


Fig. 5. A condition that a constraint group is damaged

Next, we can derive $Download_{min}$ using $Damage_{min}$. As shown in **Fig. 5**, suppose that a client downloads $Download_R$ of the total number of parity symbols and an attacker corrupts a part of the downloaded parity symbols during update. Let H_i be the event that the ith updated constraint group is recovered from the attack. If all d parity symbols of the ith constraint group are in the damaged area, $d + 1$ symbols (d parity symbols with a single updated data symbol) in this constraint group will be corrupted, and this constraint group cannot be recovered. However, if not every parity symbol is damaged as the jth constraint group, the constraint group can be recovered. Therefore, we have:

$$Pr(H_i) = 1 - \frac{\binom{Damage_R \cdot p}{d}}{\binom{Download_R \cdot p}{d}}. \tag{7}$$

Let g be the number of constraint groups to be updated and $A_{recovery}$ be the event that all the g updated constraint groups are recovered. We assume that H_i and H_j for $i \neq j$ are mutually independent. Then, the probability that all the g constraint groups are recovered from a damage is represented by the following formula:

$$Pr(A_{recovery}) = \prod_{i=1}^{g} Pr(H_i) = \left[1 - \frac{\binom{Damage_R \cdot p}{d}}{\binom{Download_R \cdot p}{d}} \right]^g . \tag{8}$$

Then, the probability of attacking at least one constraint group is

$$Pr(\overline{A_{recovery}}) = 1 - Pr(A_{recovery}) = 1 - \left[1 - \frac{\binom{Damage_R \cdot p}{d}}{\binom{Download_R \cdot p}{d}} \right]^g . \tag{9}$$

According to Definition 2, δ-robustness is guaranteed when the following formula is satisfied:

$$Pr(\overline{A_{recovery}}) = 1 - Pr(A_{recovery}) \leq \sigma . \tag{10}$$

By combining formulae (9) and (10), the following formula must be satisfied to guarantee δ-robustness:

$$Pr(\overline{A_{recovery}}) = 1 - \left[1 - \frac{\binom{Damage_R \cdot p}{d}}{\binom{Download_R \cdot p}{d}} \right]^g \leq \sigma . \tag{11}$$

If parameters p, d and σ are fixed by initial setting of a cloud system and parameter g is given by the amount of update, formula (11) varies with $Damage_R$ and $Download_R$. As $Damage_R$ increases and $Download_R$ decreases, $Pr(\overline{A_{recovery}})$ increases. For efficiency, we want to set $Pr(\overline{A_{recovery}})$ to be the maximum, i.e., $Pr(\overline{A_{recovery}}) = \sigma$.

Meanwhile, $Damage_R$ does not have to be greater than $Damage_{min}$ because the damage can be detected. The meaningful $Damage_R$ value is less than or equal to $Damage_{min}$. Therefore, the maximum $Damage_R$ in the formula is set to $Damage_{min}$. Recall $Damage_{min}$ can be calculated using formula (6). What remains is to minimize $Download_R$. As we mentioned in Example 4, the minimum $Download_R$ must be greater than $Damage_{min}$. The minimum $Download_R$ can be achieved when $Pr(\overline{A_{recovery}})$ is maximized. According to formula (11), $max\{Pr(\overline{A_{recovery}})\}$ is σ. Therefore, $Download_{min}$ can be found using the following formula:

$$max\{Pr(\overline{A_{recovery}})\} = 1 - \left[1 - \frac{\binom{Damage_{min} \cdot p}{d}}{\binom{Download_{min} \cdot p}{d}} \right]^g = \sigma . \tag{12}$$

The $Download_{min}$ value in the above formula can be computed using numerical methods. The following example is to compare the number of downloading parity symbols to ensure δ-robustness between VLCG and our scheme.

Example 5: Suppose all the parameters are the same as in Example 4, i.e., the file size is 128000 * 4 KB, encoded using a (140, 128) RS code, and the update size is one symbol. Because VLCG downloads all the parity data, if an attacker corrupts $Damage_{min} * p$ = 2,703 symbols, as in Example 4, the probability that all 12 parity symbols in the updated constraint group are damaged such that the group cannot be recovered is $1-\left[1-\binom{Damage_R \cdot p}{d}\Big/\binom{p}{d}\right]^g = 7.65\times 10^{-52}$.

This result is much smaller than σ (=10^{-10} [7]). In contrast, in the proposed scheme, it is enough to download only 18,432 symbols according to the calculated $Download_{min}$ using formula (12). This result shows that in this example, δ-robustness can be ensured by downloading fewer than the total number of parity symbols.

3.3 Algorithms for Downloading Part of Parity Blocks

Because our scheme is based on the VLCG scheme, we restrict our description only to the procedure of deciding which part of the parity blocks should be downloaded. The method of update operations itself is the same as VLCG. Additional notations to be used in our algorithms are summarized as follows:

K is a secret key.

B is a set of updated symbols.

CG is a set of the IDs of the updated constraint groups. For example, if an updated symbol is located in the jth constraint group, the ID of the constraint group is *j*.

$Block_{size}$ is the number of symbols per block.

$Block_{min}$ is the minimum number of parity blocks to be downloaded.

SL is a set of parity symbol indices requred to be updated before permutation.

$SL_{permuted}$ is a set of parity symbol indices requred to be updated after permutation.

BL is a set of block indices in which the updated parity symbols are located.

BAddL is a set of block indices to be downloaded additionally if required.

BDownL is a set of block indices required to be downloaded.

An Enhanced Remote Data Checking Scheme is a collection of the following four polynomial-time algorithms:

Algorithm 1: ***CG*** ← **DecideGroups(*B*, *K*)**

/* Input *B* consists of b_i, $1 \le i \le m$, i.e., $B = \{b_1, b_2, ..., b_m\}$. */

1: $CG = \phi$ // initialize *CG* to be empty

2: for *i*=1 to *m*

3: $CG = CG \cup \{h_K(b_i)\}$ // $h_K(b_i)$ is a cryptographic hash function

4: return *CG*

Algorithm 1 decides the constraint groups that symbols in *B* belong to using a cryptographic

hash function $h_K(b)$. The set of constraint group *CG* is initialized to be empty (Step 1). For each symbol b_i in *B*, the constraint group ID that b_i is assigned to is determined by $h_K(b_i)$ and added to *CG* (Step 3). For example, suppose the updated symbols are 1 and 5 (i.e., *B*={1, 5}) in Fig. 3. Both symbols in *B* are assigned to constraint groups 2 and 4 by $h_K(1)=2$ and $h_K(5)=4$, respectively. As a result, *CG*={2, 4}.

Algorithm 2: $\boldsymbol{Block_{min}} \leftarrow$ **Decide**$\boldsymbol{Block_{min}(Damage_{min}, g, p, d, \sigma, Block_{size})}$

/* g is calculated from the output of Algorithm 1 as $g=|CG|$ where $|CG|$ is the number of elements in *CG*. */

1: Compute $Download_{min}$ using formula (12) on $Damage_{min}$, g, p, d, σ

2: $Block_{min} = \lceil Download_{min} \times p / Block_{size} \rceil$

3: return $Block_{min}$

Algorithm 2 calculates the minimum download ratio of parity symbols $Download_{min}$ to ensure δ-robustness using formula (12), and the minimum number of downloaded parity blocks $Block_{min}$, respectively. In Step 2, the number of blocks for the minimum downloaded parity symbols is calculated because our scheme is based on parity blocks as shown in **Fig. 4** and accesing server disk is a block operation. In this algorithm, p, d, σ, $Block_{size}$ are input parameters which are already given. $Damage_{min}$ is precalculated using formula (6), and g is calculated from the output of Algorithm 1.

Algorithm 3: $\boldsymbol{BL} \leftarrow$ **CalculateParityAndBlockLocations(***CG, d, K,* $\boldsymbol{Block_{size}}$**)**

1: $SL=\phi, SL_{permuted}=\phi, BL=\phi$ // initialize *SL*, $SL_{permuted}$ and *BL* to be empty

2: for each $i \in CG$

3: for *j*=1 to *d*

4: $SL = SL \cup \{(i-1)\times d + j\}$ // Assume index starts from one

5: for each $i \in SL$

6: $SL_{permuted} = SL_{permuted} \cup \{\psi_K(i)\}$ // $\psi_K(i)$ is a pseudo-random permutation

7: for each $i \in SL_{permuted}$

8: $BL = BL \cup \{\lceil i/Block_{size} \rceil\}$

9: return BL

Algorithm 3 calculates the locations of the parity symbols after applying PRP ψ for the parity symbols in the update constraint groups that were decided in Algorithm 1. It also calculates the indices of blocks that the permuted parity symbols are located in the server. Let us continue the previous example using **Fig. 3**. Since *CG*={2,4} and *d*=2, *P3* and *P4* (or *P7* and *P8*) are parity symbols in the constraint group 2 (or 4), respectively. Thus, *SL*={3,4,7,8} (Step 2~4). In **Fig. 3**, *P3*, *P4*, *P7* and *P8* move to the 6th, 7th, 5th and 8th

positions by PRP ψ, respectively. Thus, $SL_{permuted}$={5,6,7,8} (Step 5~6). In **Fig. 4**, we set $Block_{size}$=2, Thus, BL={3,4} (Step 7~8).

Algorithm 4: ***BDownL*** ← **GenerateDownloadBlocks(*$Block_{min}$*, *BL*, *K*)**

1: $BAddL = \phi$ // initialize *BAddL* to be empty

2: if $Block_{min} > |BL|$ // $|BL|$ is the number of elements in *BL*

3: while $|BL \cup BAddL| \le Block_{min}$

4: $BAddL = BAddL \cup \{\pi_K\}$ /* π_K is a pseudo-random function that generates an integer in the range from one to $\lceil p / Block_{size} \rceil$. */

5: $BDownL = BL \cup BAddL$

6: return $BDownL$

If the number of blocks in *BL* that was calculated in Algorithm 3 is less than $Block_{min}$, Algorithm 4 generates additional blocks *BAddL* to meet the minimum number of downloaded parity blocks using a pseudo-random function (PRF) π_K, and integrates two arrays of the blocks, *BL* and *BAddL* into *BDownL*.

3.4 δ-robustness Guarantee

In Section 3.2, we demonstrated that when the download ratio of parity data is more than $Download_{min}$, δ-robustness can be ensured. Therefore, the following theorem holds:

Theorem 1: Our scheme guarantees δ-robustness.

Proof: In order to guarantee δ-robustness, both conditions of Definition 2 must be satisfied. To satisfy the first condition of Definition 2, formula (4) should hold. The $Damage_{min}$ that is calculated by formula (6) is the minimum value of $Damage_R$ which satisfies formula (4). If an attacker corrupts more than $Damage_{min}$ fraction of $\tilde{F}$, our scheme can detect the corruption with high probability (more than 1-σ). Thus, our scheme satisfies the first condition of Definition 2.

To satisfy the second condition of Definition 2, formula (11) should hold. The $Download_{min}$ that is calculated by formula (12) is the minimum value of $Download_R$ which satisfies formula (11). Since our scheme downloads at least $Download_{min}$ fraction of $\tilde{F}$, it satisfies the second condition of Definition 2. □

Next, we show that the value of δ in our scheme is the same as that in VLCG. We define $detect_{min}$ as the minimum ratio of the corruption in the parity data in the scope in which the corruption is detectable, and $recover_{max}$ as the maximum ratio of the corruption in the parity data in the scope in which the corruption is recoverable. In **Fig. 2(a)**, when *Pr*(Detect) approaches one, the corresponding value of the horizontal axis, i.e., the ratio of corrupted data, is set as $detect_{min}$; whereas when *Pr*(Recovery) begins to decline from initial value one,

the value of horizontal axis at that moment is set as $recover_{max}$. Thus, $detect_{min}$ and $recover_{max}$ are determined about 0.01 and 0.04, respectively in **Fig. 2(a)**. If a scheme ensures δ-robustness, the δ can be any value between $detect_{min}$ and $recover_{max}$.

In Section 3.2, we showed that if the corrupted ratio of the parity data is more than $Damage_{min}$, the corruption can be detected. Thus, we can have $detect_{min} = Damage_{min}$. According to formula (12), it is recoverable even when the ratio of corruption in the parity data is equal to $Damage_{min}$. Thus, it holds that $recover_{max} \geq detect_{min}$. This relationship between $detect_{min}$ and $recover_{max}$ is also shown in **Fig. 2(a)**. Even if the δ can be any value between $detect_{min}$ and $recover_{max}$, we set the δ as $\delta = min(detect_{min}, recover_{max}) = detect_{min} = Damage_{min}$. Because all of the parameters for calculating $Damage_{min}$ in VLCG and in our scheme are the same, the value of δ in our scheme is the same as that in VLCG.

3.5 Summary and Comparison

In VLCG, the bandwidth factor is $\alpha = \frac{|P|}{|D|}$ because the client needs to download all of the parity data in the Update phase. In contrast, in the proposed scheme, the client need not download all of the parity data for small updates; hence, the bandwidth factor is $\alpha = \frac{Download_R \cdot |P|}{|D|}$, $(0 < Download_R \leq 1)$, which is affected by $Download_R$. The result of $Download_R$ is shown in the next section, as shown in **Figs. 6** and **7**. For example, in a condition where the file size is 256 MB and the block size is 4 KB, $Download_R = 0.18$, while the update size is 100 symbols. The performance and security analysis results can be summarized as in **Table 1**, which implies that the proposed scheme enhances efficiency without security degradation with regard to δ-robustness.

Table 1. Comparison of the performance among πR-D, VLCG, and our scheme

	Probability of detection	Update bandwidth	Robustness
πR-D	$1-(1-f_{cb})^C$	$\alpha = \frac{\lvert F\rvert}{\lvert D\rvert}$ (insert/delete) $\alpha = \frac{\lvert P\rvert}{\lvert D\rvert}$ (modify)	δ-robustness
VLCG	$1-(1-f_{cb})^C$	$\alpha = \frac{\lvert P\rvert}{\lvert D\rvert}$	δ-robustness
Our scheme	$1-(1-f_{cb})^C$	$\alpha = \frac{Download_R \cdot \lvert P\rvert}{\lvert D\rvert}$ $(0 < Download_R \leq 1)$	δ-robustness

(f_{cb} : the fraction of the corrupted block, C: the number of symbols checked by spot checking)

4. Experimental Results

We conducted two experiments. In both of the experiments, the blocks are encoded with a (140, 128) RS code and the block size in the server is 4 KB. The first experiment is to test the efficiency of the communication overhead for varying parameters. The proposed scheme reduces communication overhead for varying file sizes, owing to the partial download, as compared to the VLCG scheme.

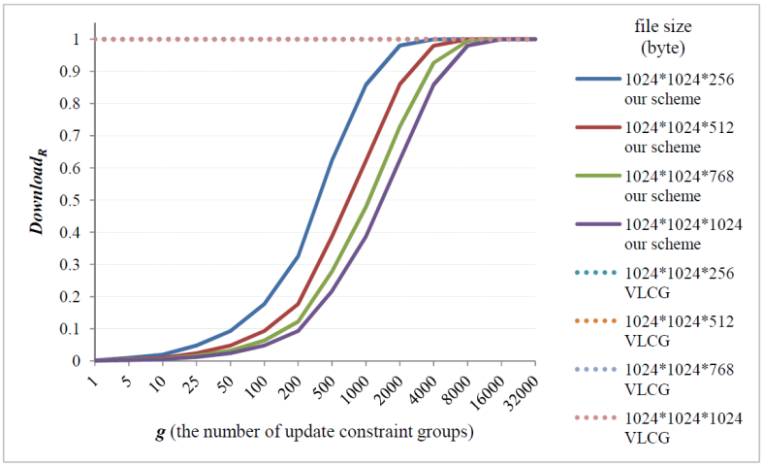


Fig. 6. Communication overhead of the enhanced remote data checking scheme

As shown in **Fig. 6**, in the VLCG scheme, since all the parity data should be downloaded regardless of the file size and the number of updated constraint groups, the four dotted lines for $Download_R$ are coincident, and the value of $Download_R$ is always one, irrespective of the change in parameters. Thus, all the parity data should be downloaded for any small update in the VLCG scheme. However, in the proposed scheme, only the parity symbols in the blocks the updated parity symbols belong to are downloaded. Thus, when the file sizes are determined, the greater the number of updated constraint groups, the greater the number of parity data that need to be downloaded. Generally, the larger the file size, the greater the number of constraint groups. Thus, if the number of updated constraint groups is fixed, the bigger the file size, the less the ratio of the number of blocks that need to be downloaded to the total number of parity blocks. It needs smaller communication overhead, i.e., $Download_R$, in a larger file size. Through this experiment we can obtain the following result: when the size of updated data is small, because of the partial download of parity data, the communication overhead for update in the proposed scheme is less than that in the VLCG scheme.

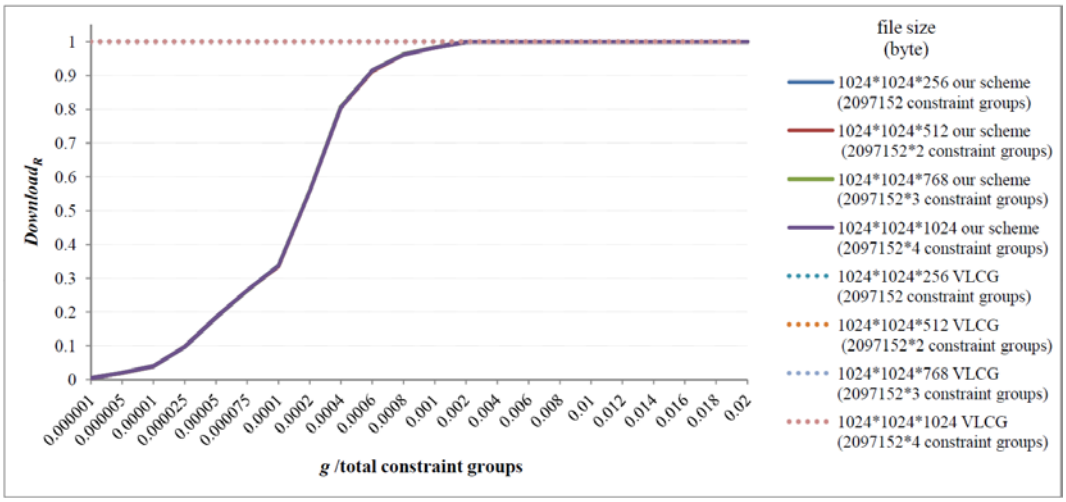


Fig. 7. Communication overhead for normalized g

The result of the VLCG scheme in **Fig. 7** is the same as that in **Fig. 6**, the four dotted lines for $Download_R$ are coincident, and the value of $Download_R$ is always one. Through the result in **Fig. 6**, we can observe that in the proposed scheme, the ratio of downloaded parity data depends on the amount of the updated data and the file size. The communication overhead for update becomes smaller as the update size becomes small and the file size becomes large. However, as shown in **Fig. 7**, if the number of updated constraint groups is normalized to the total constraint groups, the communication overhead for varying file sizes is the same as well. Moreover, when the ratio of the number of updated constraint groups to the number of total constraint groups is small, the value of $Download_R$ is less than one. Thus, the efficiency of the proposed scheme depends on the proportion of updated symbols to the file size. Through this proportion we can determine whether the communication overhead in the proposed scheme is less than that in the VLCG scheme.

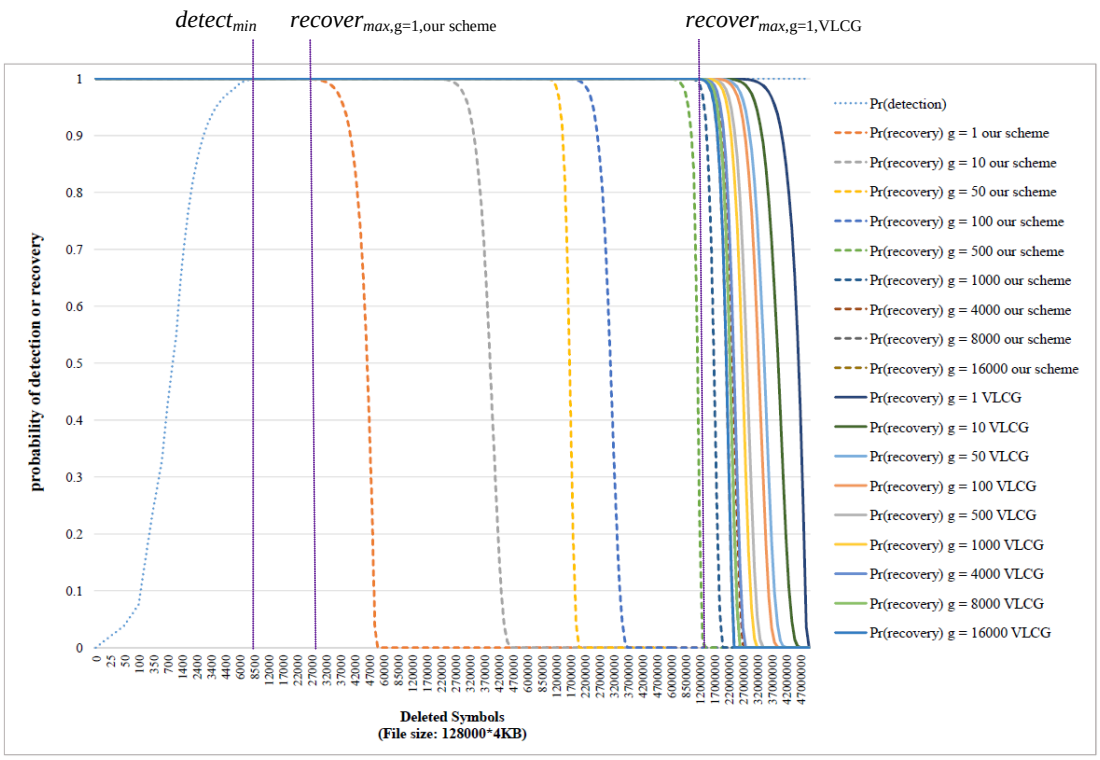


Fig. 8. δ-robustness guarantee

The second experiment is to test whether our scheme ensures δ-robustness and to compare our scheme and the VLCG scheme in terms of the probability of detection or recovery. In **Fig. 8**, there is only one line for the probability of detecting corruption, because it is the same in both schemes. Recall our scheme uses the same formula, i.e., formula (2), to calculate the probability of detecting corruption as VLCG. In contrast, the lines for the probability of recovery are numerous because they vary with different g values and schemes. When Pr(detection) approaches one, we set the value of the horizontal axis, i.e., the ratio of deleted blocks over the total parity blocks as $detect_{min}$, whereas when Pr(recovery) has held one but begins to decline, we set the value of horizontal axis as $recover_{max}$. We can observe $recover_{max} \geq detect_{min}$ from **Fig. 8**, as from Section 3.4. In the figure, although two $recover_{max}$

($recover_{max,g=1,\text{our scheme}}$ and $recover_{max,g=1,\text{VLCG}}$) for only g=1 are shown, there must be two $recover_{max}$ for each g.

There is an overlapped interval in the horizontal axis in which both Pr(detection) and Pr(recovery) are one between $detect_{min}$ and $recover_{max}$. According to **Fig. 2**, both schemes ensure δ-robustness. Any value of the horizontal axis in the overlapped interval can be δ. In Section 3.4, the minimum value in the overlapped interval is set to δ and is the same as $detect_{min} = min(detect_{min}, recover_{max})$. This δ value is the same in both VLCG and our scheme. The overlapped interval is shorter in our scheme than in VLCG. Actually, the amount of downloading parity data is affected by $recover_{max}$. As $recover_{max}$ is smaller, the amount of downloading parity data decreases. Because our scheme has smaller $recover_{max}$ than VLCG, it has smaller communication overhead. In the figure, the smaller the g value, the shorter the overlapped interval. Therefore, our scheme is more efficient for small g values as it was claimed in Section 3.1.

Summarizing the second experiment, when an attacker corrupts a maximum of δ-fraction of the parity data, the client can recover the data with high probability. When the attacker corrupts more than a δ-fraction of the parity data, the client can detect the attack with high probability. Therefore, both schemes can ensure δ-robustness. The only difference between the two schemes is that the value of $recover_{max}$ in our scheme is less than that of the VLCG scheme. The $recover_{max}$ affects the amount of downloading parity data. Therefore, we could calculate $Download_{min}$ in Section 3.2, whereas it is the total parity data in VLCG. Our scheme is more efficient for small g values.

5. Conclusion and Future Work

In the VLCG scheme, all of the parity data must be downloaded for update operations to ensure δ-robustness. In this paper, we proposed an Enhanced Remote Data Checking Scheme to reduce the communication overhead for update operations. In the proposed scheme, the client downloads only part of the parity data for update operations. The downloaded parity data include not only the parity data of the updated constraint groups but some redundant parts of the parity data as well to avoid discovering the association between symbols and constraint groups.

To include some redundancy, the client downloads all symbols in the blocks in which the updated parity symbols are located. However, if the number of updated symbols is small, this redundancy is insufficient to hide the association. Therefore, we presented a method to add more redundancy. By calculating the minimum download ratio of the parity symbols, the proposed scheme can ensure δ-robustness in the condition of downloading only a part of the parity data. If the number of all symbols in the blocks in which the updated parity symbols are located is less than the minimum download ratio of the total parity symbols, we select some additional blocks randomly among all the parity blocks by a pseudorandom function.

In addition, we found that the δ value in our scheme is the same as $Damage_{min}$, which is the δ value in VLCG as well. This means that our scheme maintains the same security level as VLCG while it downloads fewer parity data. To summarize, the proposed scheme reduces the communication overhead for update operations without any security degradation.

From the efficiency point of view, our experiment showed that it requires less communication overhead when the update size is smaller relative to the file size. From the security point of view, our experiment showed that the proposed scheme can ensure δ-robustness for varying parameters. A graph based on the experiment showed that there is an

overlapped interval in the horizontal axis in which both *Pr*(detection) and *Pr*(recovery) are one. This means that the proposed scheme guarantees δ-robustness. In addition, the graph showed that the δ value ($detect_{min}$) in the proposed scheme is the same as the value of δ in the VLCG scheme. The only difference between the two schemes is that the value of $recover_{max}$ in our scheme is less than that of the VLCG scheme, because the downloaded ratio of parity data in VLCG is more compared to our scheme.

In the proposed scheme, the client plays the role of an auditor. It is possible to permit a third-party auditor (TPA), trusted by both the server and client, to check the integrity of the outsourced data [22], [23]. In future work, it may be interesting to incorporate the TPA technique into our scheme for the purpose of public auditing. We considered dynamic updates only for a single client / server. However, files in multiple servers can be accessed by multiple clients. Reducing communication overhead for dynamic updates in the multiple client / server environment is also intended for future work.

Acknowledgement

This research was supported by the Chung-Ang University Research Scholarship Grants and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2013R1A2A2A01005559 and 2010-0022851) in 2013.

References

[1] H. Takabi, J. Joshi and G. Ahn, "Security and Privacy Challenges in Cloud Computing Environments," *IEEE Security and Privacy*, vol. 8, no. 6, pp. 24-31, November, 2010. Article (CrossRef Link)

[2] Z. Xiao and Y. Xiao, "Security and Privacy in Cloud Computing," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 843-859, July, 2013. Article (CrossRef Link)

[3] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson and D. Song, "Provable data possession at untrusted stores," in *Proc. of 14th ACM Conference on Computer and Communications Security*, pp. 598-609, 2007. Article (CrossRef Link)

[4] A. Juels and B. S. Kaliski, "PORs: Proofs of retrievability for large files," in *Proc. of the 14th ACM Conference on Computer and Communications Security*, pp. 584-597, 2007. Article (CrossRef Link)

[5] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. of Asiacrypt*, vol 5350 of LNCS, pp. 90-107, December, 2008 Article (CrossRef Link)

[6] K. D. Bowers, A. Juels and A. Oprea, "HAIL : a high-availability and integrity layer for cloud storage," in *Proc. of 16th ACM Conference on Computer and Communications Security*, pp. 187-198, 2009. Article (CrossRef Link)

[7] C. Erway, A. Kupcu, C. Papamanthou and R. Tamassia, "Dynamic provable data possession," in *Proc. of 16th ACM Conference on Computer and Communications Security*, pp. 213-222, 2009. Article (CrossRef Link)

[8] M. T. Goodrich and R. Tamassia, "Efficient authenticated dictionaries with skip lists and commutative hashing," *Johns Hopkins Information*, January, 2001. Article (CrossRef Link)

[9] M. T. Goodrich, R. Tamassia and A. Schwerin, "Implementation of an authenticated dictionary with skip lists and commutative hashing," in *Proc. of DARPA Information Survivability Conference & Exposition II*, pp. 68-82, 2001. Article (CrossRef Link)

[10] C. Papamanthou, R. Tamassia and N. Triandopoulos, "Authenticated hash tables," in *Proc. of 15th ACM Conference on Computer and Communications Security*, pp. 437-448, 2008. Article (CrossRef Link)

[11] G. Ateniese, R. D. Pietro, L. V. Mancini and G. Tsudik, "Scalable and efficient provable data possession," in *Proc. of the 4th International Conference on Security and Privacy in Communication*, Article No. 9, 2008. Article (CrossRef Link)

[12] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson and D. Song, "Remote data checking using provable data possession," *ACM Transactions on Information and System Security*, vol. 14, no. 1, Article No. 12, 14 May, 2011. Article (CrossRef Link)

[13] Wikipedia, http://en.wikipedia.org/wiki/Forward_error_correction

[14] R. Curtmola, O. Khan and R. Burns, "Robust remote data checking," in *Proc. of 4th ACM International Workshop on Storage Security and Survivability*, pp. 63-68, 2008. Article (CrossRef Link)

[15] K. D. Bowers, A. Juels and A. Oprea, "Proofs of retrievability: Theory and implementation," in *Proc. of the 2009 ACM Workshop on Cloud Computing Security*, pp. 43-54, 2009. Article (CrossRef Link)

[16] B. Chen and R. Curtmola, "Robust dynamic provable data Possession," in *Proc. of 32nd International Conference on Distributed Computing Systems Workshops*, pp. 515-525, 2012. Article (CrossRef Link)

[17] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300-304, 1960. Article (CrossRef Link)

[18] C. Hanser and D. Slamanig, "Efficient Simultaneous Privately and Publicly Verifiable Robust Provable Data Possession from Elliptic Curves," in *Proc. of International Conference on Security and Cryptography*, pp. 15-26, 2013. Article (CrossRef Link)

[19] J. S. Plank, S. Simmerman and C. D. Schuman, Jerasure: A library in C/C++ facilitating erasure coding for storage applications – Version 1.2, University of Tennessee, Tech. Rep. CS-08-627, 2008. Article (CrossRef Link)

[20] J. Katz and Y. Lindell, Introduction to Modern Cryptography, Chapman & Hall/CRC, 2008.

[21] Wikipedia, http://en.wikipedia.org/wiki/Cryptographic_hash_function

[22] C. Wang, Q. Wang, K. Ren and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *Proc. of IEEE INFOCOM*, pp. 1-9, March 2010. Article (CrossRef Link)

[23] Q. Wang, C. Wang, J. Li, K. Ren and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *Proc. of the 14th European Conference on Research in Computer Security*, pp. 355-370, 2009. Article (CrossRef Link)

Lin Dong received the BS degree from Huaiyin Normal University, China, and Konkuk University, Seoul, Korea in 2012 and the MS degree in Electronics and Electrical Engineering from Chung-Ang University, Seoul, Korea in 2014. Her research interests include cryptology and cloud computing security.

Jinwoo Park received the BS degree from Chung-Ang University, Seoul, Korea in 2013. He is currently pursuing his MS degree in School of Electrical and Electronics Engineering, Chung-Ang University, Seoul, Korea. His research interests include big data processing, data analytics and internet security.

Junbeom Hur received the B.S. degree in computer science from Korea University in 2001, the M.S. and Ph.D. degrees in computer science from KAIST in 2005 and 2009, respectively. He has been in the University of Illinois at Urbana-Champaign as a postdoctoral researcher from 2009 to 2011. He is currently an assistant professor in the school of computer science and engineering at the Chung-Ang University in Korea. His research interests include information security, mobile computing security, cyber security, and applied cryptography.

Ho-Hyun Park received the BS degree from Seoul National University, Seoul, Korea in 1987 and the MS and Ph.D. degrees from KAIST in 1995 and 2001, respectively. From 1987 to 2002, he worked at Samsung Electronics as a principal engineer. He is currently a professor in School of Electronics and Electrical Engineering, Chung-Ang University, Seoul, Korea. His research interests include multimedia processing, big data and internet security.