# Trapdoor Digital Shredder: A New Technique for Improved Data Security without Cryptographic Encryption

**Taek-Young Youn[1] and Nam-Su Jho[1*]**
[1] Electronics and Telecommunications Research Institute
138 Gajeongno, Yuseong-gu, Daejeon, Korea
[e-mail: taekyoung, nsjho@etri.re.kr]
*Corresponding author: Nam-Su Jho

*Received January 22, 2019; revised April 2, 2019; accepted July 28, 2019; published March 31, 2020*

## *Abstract*

Along with the increase of the importance of information used in practice, adversaries tried to take valuable information in diverse ways. The simple and fundamental solution is to encrypt the whole data. Since the cost of encryption is increasing along with the size of data, the cost for securing the data is a burden to a system where the size of the data is not small. For the reason, in some applications where huge data are used for service, service providers do not use any encryption scheme for higher security, which could be a source of trouble. In this work, we introduce a new type of data securing technique named *Trapdoor Digital Shredder*(TDS) which disintegrates a data to multiple pieces to make it hard to re-construct the original data except the owner of the file who holds some secret keys. The main contribution of the technique is to increase the difficulty in obtaining private information even if an adversary obtains some shredded pieces. To prove the security of our scheme, we first introduce a new security model so called IND-CDA to examine the indistinguishability of shredded pieces. Then, we show that our scheme is secure under IND-CDA model, which implies that an adversary cannot distinguish a subset of shreds of a file from a set of random shreds.

***Keywords:*** Trapdoor digital shredder, Data privacy, Encryption

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2017-0-00213, Development of Cyber Self Mutation Technologies for Proactive Cyber Defence)

## 1. Introduction

In these days, to protect someone's data is one of fundamental requirements for many information services. So, various techniques have been introduced to protect significant information from adversarial trials throughout all steps of data managements including publication [14], processing of personal data [13,15], and data storing [2,6,10]. However, the security against an adversary who obtain partial stored data in a remote storage was not considered in depth since we can counter the adversary by preventing the adversary from accessing to the target data or by using well-known encryption schemes such as AES [1] to encrypt data as in [2]. The former has some demerits in the sense that access-control based countermeasures are broken in various ways. Hence, until now, it is widely believed that the use of encryption schemes is the most reliable way for securing data since we can prevent the adversary from extracting meaningful information from encrypted data due to the security of the underlying encryption scheme.

We already have many standardized tools for data encryption. However, many users do not use the schemes to encrypt their data due to the following reasons. The first reason is the cost of operation. The cost of encryption is not a burden when the volume of data is small, but we cannot ignore the cost of encryption when the volume of data is huge. Since the volume of data is huge in these days and the size of files is increasing very fast, to encrypting all data is a burden for some users. To overcome the performance issue, many researchers have tried to improve the efficiency of standard encryption schemes [3]. In these days, AES implimentation technique for both SW and HW are almost optimized along with recent researches, but the cost of encryption scheme is still a burden which can not be ignored. So, it is desirable to design a technique that can pretect data privacy with lower cost than AES. Another reason is the inconvenience caused by the use of encryption schemes. It is not desirable to use encryption schemes for some data which are frequently modified since the owner of a file should decrypt encrypted file and re-encrypt the file whenever he wants to modify the file.

On the other hands, many researches have tried to improve the data privacy by adopting multiple storage servers and distributing a file to the servers [2,6,10]. Though existing techniques can improve the security of stored data using distributed storage servers, they still require costly operations to encrypt stored data using encryption schemes [2] or to generate encoded file using cryptographic tools such as secret sharing or information dispersal algorithm [6,10]. Before to discuss the main idea of this paper, we want to emphasis that the goal of this work is not support the same security that can be supported by conventional security algorithms such as DES and AES. Instead of giving the same security, this work is made to support sufficient security without using traditional security algorithms which require costly operation for large files.

Basically, we use two ideas to design our scheme. The first one is the shredding of the target data and the second idea is the use of distributed storage servers for shredded data. Until now, various techniques have been studied to improve the security of stored data. However, among two basic features of our scheme, the first idea is not widely considered for securing data since it is believed that the approach cannot prevent an adversary from obtaining meaningful information from shreds. In practice, it may be possible to re-construct the original file from the shreds if the adversary has all of shreds. However, it can be a good candidate for securing data by combining the first idea with the second idea. In other words, we can improve the security of a file by shredding the file and storing shreds in a distributed way since an adversary who breaks a storage server only can obtain a part of the file. However, the use of distributed storage servers for higher security is already widely considered in the literature. In this viewpoint, a distinguishing characteristic of our construction is the randomness of data stored in storage servers. Specifically, in our construction, we randomize a subset of shreds which will be stored in the same storage so that an adversary can obtain a random shreds of a file even if the adversary succeeds in breaking the storage server. In the adversary's viewpoint, a shred of a random position of the original file does not give meaningful information. One more point that should be considered is the size of a shred. To prevent the adversary can obtain any meaning information from a shred, the size of a shred is determined so that any meaningful information cannot be expressed in a shred. Based on the above described idea, in this paper, we design a new technique to improve data security without cryptographic encryption.

The contributions of this paper can be summarized as following:

- *Novel approach for securing data.* We propose a new and novel method which can improve the security of data which are stored in a remote storage.

- *Practical solution.* Our solution does not require costly cryptographic operations, and thus we can apply our technique to applications where huge data should be securely and efficiently managed.

The remainder of this paper is organized as follows. In Section 2, we describe security models for our construction. In Section 3, we describe our technique with details, and its security and performance are discussed in Section 4. In chapter 5, we conclude this paper.

## 2. Security Models

In this section, we will discuss some security-related topics. Then, we also give formal security definition for our proposed technique.

### 2.1 Attack Environment

Recall that the trapdoor digital shredding technique is designed to guarantee the security of data which are stored in remote storage servers. Specifically, we consider the environment that there are multiple storage servers which operate independently and data are distributed to the servers. Therefore, an adversary can access limited partial data even if some of the servers collude with an adversary. Here, we ignore the scenario that the adversary obtains entire data by colluding all of storage servers at the same time. To prevent the adversary, we require more complicated cryptographic methods, usually impractical.

### 2.2 Goal of Attack

The goal of an adversary is obtaining meaningful information from stored data. Remind that we assume that an adversary can access one or more storage servers at the same time. Therefore, the goal of an adversary can be considered as obtaining meaningful information from partial data. If a server stores data containing long consecutive bit-strings of the original file, then it is too easy for the adversary to achieve the goal. Note that the number of storage servers is limited by physical constraints so that it is hard to reduce the size of stored data in each server. Instead, we need strategies for constructing partial data which are large enough but contains no long consecutive bit-string of the original file.

## 2.3 Formal Security Definition

For describing formal security definition, we define a function named *trapdoor digital shredding* (*TDS*) algorithm. *TDS* algorithm shreds the input file $F$ into small data fragments with pre-assigned bit-length $\ell$. We will denote the small data fragment as *shred*. After shredding, *TDS* divides shreds into $N$ sets, where each set composes a *chunk*. Formally,

$$TDS(F, \ell, N, tk) = \{CK_1, CK_2, \ldots, CK_N\},$$

where $tk$ means trapdoor secret key which is required to re-construct the original file from the chunks.

To define the security model for *TDS*, we consider the following game. In the game, an adversary gets chunks as input. To simplify the security model, we assume that an adversary can access one server, i.e. one chunk. Note that the security model can be extended for an adversary who can access to multiple chunks easily.

Defining the goal of adversary in the game is somewhat complicated. Note that if the goal of the adversary is distinguishing a given chunk from random string with the same size, it can be a good candidate for reflecting the security of *TDS*. However, winning of the game highly depends on the selection of the file $F$. For example, if a file of which all bits are valued with ‘0’ is selected as input for *TDS*, then it is too easy to distinguish the chunk from a random string. Therefore, we need a modified definition to mitigate the effects of single file's characteristic.

Before more discussions, we define a notion *dictionary* for a given file $F$ as a set of all shreds of $F$, including duplications.

$$Dic(F) = \{\text{set of all shreds of } F\}.$$

Now we can consider the goal of an adversary as finding the original file for a given chunk, where two files with the same dictionary are also given. The second file is made by shuffling the original file to contain no long consecutive bit-string. Note that if there is an adversary who can obtain a sufficiently long consecutive bit-string of the original file from a given chunk with high probability, then it is easy for the adversary to distinguish the two files. Therefore, proving the nonexistence of the adversary, who can win the game with high probability, implies the security of *TDS* technique.

Formally, we define IND-CDA(indistinguishability against chosen dictionary attack) games as followings. In the game, $\mathfrak{A}$ selects two files $\{F_0, F_1\}$ such that

$$Dic(F_0)= Dic(F_1)$$

and sends the files to $\mathfrak{B}$. $\mathfrak{B}$ chooses $i$ from 0 or 1, and runs $TDS(F_i, \ell, N, tk)$ with randomly selected $tk$. The adversary $\mathfrak{A}$ chooses $j$, the index of a corrupted chunk. Then $\mathfrak{B}$ sends $CK_j$ to $\mathfrak{A}$. Finally, $\mathfrak{A}$ guesses the $\mathfrak{B}$'s choice $i$, viewing $F_0$, $F_1$, and $CK_j$. If $\mathfrak{A}$'s guess is correct, then the adversary wins the game.

**Definition**(IND-CDA security) A *TDS* system satisfies IND-CDA security if there is no adversary who can win the IND-CDA game with non-negligible probability.

## 3. Our Construction

In this section, we will describe our construction. Let $F$ be the file to store and $N$ be the number of distributed storage servers. Recall that the goal of this work is to design a secure and practical way to distribute the file $F$ to $N$ servers so that an adversary cannot extract meaningful information from a chunk stored in a corrupted server. Simply, our scheme consists of two phases: the chunk generating phase and the data distribution phase. Since we are not interested in the way of distributing each chunk, the main goal of this work is to give a secure and efficient way to generate chunks for the file $F$. Hence, from now, we will describe the way to generate data chunk for digital shredding technique.

### 3.1 Basic Idea

Here we describe our basic idea before giving concrete scheme. We expect that the original file will be disintegrated into tiny shreds and re-constructed as chunks. Here, we need that each chunk should not reveal any meaningful information regarding the original file. Note that since we are not considering about applying any cryptographic method like encryption system, the chunks are just collections of tiny shreds without any modification. In other words, a shred in a chunk has exactly the same value to the corresponding shred in the original file.

Therefore, in our construction two points are important for security. The first point is the size of a shred since the size influences on the hardness of obtaining meaningful information from a single shred. Intuitively, it is hard to extract meaningful information from small shred than from larger one. Extremely, a shred has no information if the size of the shred is 1-bit. However, smaller shreds are hard to handle since the number of shreds is increased for smaller shreds. The second point for secure digital shredding is randomness of the chunk construction. If an adversary can correctly guess the original positions of consecutive shreds, he can obtain meaningful information from the shreds without regarding the size of each shred. Fortunately, it is assumed that the adversary can access to limited number of chunks so that it is hard to obtain a large fragment of the original file. However, the adversary still can guess some partial information of the file $F$ if the position of the shreds can be determined. It is why we need to randomize the positions of shreds. The basic procedure for our scheme can be seen in **Fig. 2**.

## 3.2 Trapdoor Digital Shredding System

In this section, we describe the first trapdoor digital shredding system. In the system, a client and a set of storage servers are communicate to upload and download data. For the explanation, let *Clt* be the client who wants to store a file *F* to *N* servers $\{Srv_1, Srv_2, \ldots, Srv_N\}$. In the following description, we didn't give any concrete way to identify a file or the way to manage stored file since they are not included in the scope of this work. The trapdoor digital shredding system works as follows.

- **Upload.** To upload the file *F*, the client *Clt* generates an identifier $id_F$ for the file and runs the *chunk generation function*, *C-Gen*(), to generates chunks. The function runs mainly in two steps;

  1. shred target file into tiny shreds; and
  2. construct random subsets of shreds.

  The function requires four inputs, the target file to be shredded, the size of shreds, the number of storage servers, and the trapdoor key. Note that, the trapdoor key will be used in chunk generation and re-construction of the original file from chunks. It should be hard to re-construct the original file without the trapdoor key, and it is one of fundamental security requirements. The chunk generation function can be written as a formula as following:

$$\{CK_1, CK_2, \ldots, CK_N\} = C\text{-}Gen(F, \ell, N, tk)$$

  where *tk* is the trapdoor key and $\ell$ is the size of a shred which implies that $|CK_i| = \ell$ for all *i*. Then the client sends $CK_i$ to $Srv_i$ for all $i = 1, \ldots, N$, and keeps the trapdoor key *tk* securely. Note that it is not clear yet how does the digital shredding function *C-Gen*() works since we didn't give concrete description. We will give concrete algorithms for the function with more discussions in Section 3.3.

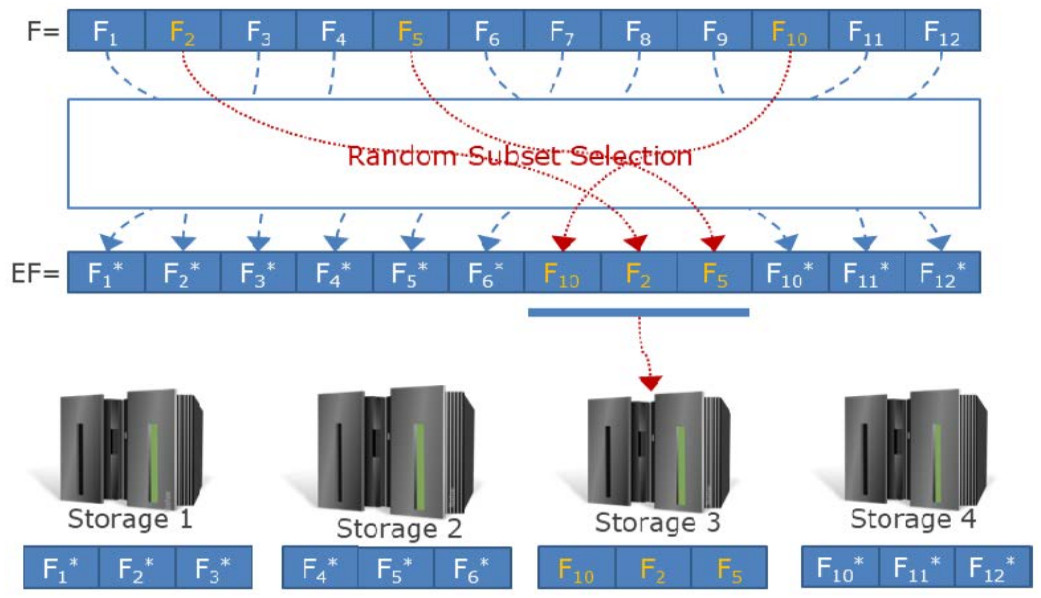


**Fig. 1.** Basic idea of our construction

- **Download.** To retrieve the file $F$, the client downloads $CK_i$ from $Srv_i$ for all $i = 1, \ldots, N$, and runs *file retrieval function*, *F-Ret*() with the private trapdoor key *tk*. To re-construct the original file, the client has to rearrange all shreds in $N$ chunks using the trapdoor key. The file retrieval function can be written as a formula as following:

$$F = F\text{-}Ret(\ CK_1, \ldots, CK_N, tk).$$

Concrete description for the re-construction procedure will be given in Section 3.4. Note that it is possible to design *F-Ret* to re-construct the original file from partial collection of $N$ chunks (if necessary) by applying error correcting code.

The proposed system composed with two functions as explained in the above, and the *C-Gen* function, which shreds the file and constructs chunks, is a core algorithm for the system. At glance, it resembles a paper shredding machine in the sense that a paper is shredded into tiny shreds. The purpose of the shredding machine is making it impossible to rearrange the original paper from a dump of shreds. We also make the rearrangement harder using multiple dumps. This approach is main difference between our scheme and ordinary data distribution techniques, where a file is simply divided into multiple chunks and stored in a distributed way. Note that, in ordinary data distribution techniques, a chunk is usually a large loaf of the original file.

## 3.3 How Implement Random Subset Selection

As we explained in the above, the design of the chunk generation function *C-Gen*() is the most important part for the proposed system. Here, we give a concrete example of the function. Recall that the function is executed when a client *Clt* uploads a file $F$. The first step of *C-Gen* is to generate an encoded file *EF* from the file $F$. To encode the file, *Clt* chooses a random trapdoor key *tk* and generates $len_1$-bit random string which can be represented as following:

$$n_1 \,||\, j_1 \,||\, n_2 \,||\, j_2 \,||\, \cdots \,||\, n_k \,||\, j_k = prng(len_1, tk),$$

where *prng*(*len*, *seed*) is a pseudo random function which returns *len*-bit random number based on *seed*. In the above equation, $n_i$ and $j_i$ mean the $i$-th noise-value and noise-interval, respectively. For the efficiency of noise management, it is better to use word-sized noise-value in practice. However, we use random-sized noise-value to randomize the position of each shred. To support random-sized noises, we will use some part of $n_i$ to decide the size of the $i$-th noise, but we use the above notation in the current form for convenience. The noises are inserted in the file $F$ as follows. The first noise $n_1$ is inserted between the $j_1$-th bit and the ($j_1$+1)-th bit of $F$. The second noise $n_2$ is inserted between the ($j_1$+$j_2$)-th bit and the ($j_1$+$j_2$+1)-th bit of $F$, which means that the interval from the first noise to the second noise is $j_2$ bits. Similarly, the position of the $i$-th noise $n_i$ is $j_i$-bits far from the end of the ($i$-1)-th noise. In **Fig. 2**, we give an example for the noise insertion procedure for $k = 5$.

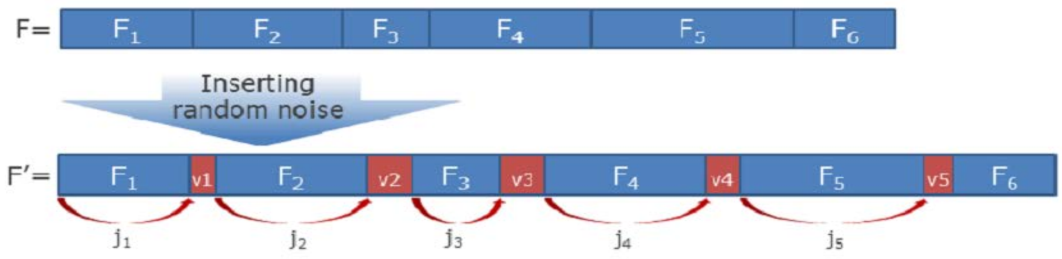


**Fig. 2.** Data processing procedure (noise insertion)

As seen in the **Fig. 2** random noises are distributed in $F$ at the random positions. Let $EF$ be the encoded file with random noises. The next step is to shred the encoded file. Since we assumed that there are $N$ servers, the file $EF$ is divided into $tN$ shreds for an integer $t$. If $\ell_{EF}$ is the length of the file $EF$, we need $\ell_{EF} = \ell tN$. Remind that the size of each shred is $\ell$. Therefore, we append redundant bits to $EF$ if necessary. In this paper, we assume that $\ell_{EF} = \ell tN$ for convenience. Then the encoded file $EF$ is shredded into $tN$ shreds. Let $P_i$ be the $i$-th shred of $EF$. We construct the $N$ subsets as $S_j = \{P_i \mid i = j \bmod N\}$. At this time, we have $N$ subsets of the encoded file $EF$, and each subset is a collection of random shreds of the original file $F$. Note that, without noises, positions of shreds in $S_j$ can be easily predicted.

The last step of chunk generation is shuffling each $S_j$ randomly. Note that each $S_j$ consists of $t$ elements. Therefore, we explain the shuffling technique with $S_j = \{P_1^{(j)}, P_2^{(j)}, \ldots, P_t^{(j)}\}$. First, we need to compute

$$p[1] \parallel p[2] \parallel \cdots \parallel p[t] = prng(len_2, tk \parallel j),$$

where each $p[i]$ is an integer with $1 \leq p[i] \leq t$. For $k = 1$ to $t$, exchange $P_k^{(j)}$ with $P_{p[k]}^{(j)}$, sequentially. Then, $Clt$ sends $j$-th chunk $CK_j$, the concatenation of all shreds in $S_j$, to the $j$-th server.

### 3.4 File Retrieval

*F-Ret* function runs in reverse way of *C-Gen* function. In other words, *F-Ret* re-shuffles each chunk, gathers the chunks in one encoded file, and removes random noises sequentially to re-construct the original file. Note that only the *Clt* who keeps the random trapdoor key *tk* can generate the same random strings of the steps in the chunk generation. Therefore, *F-Ret* function can be runned correctly by the *Clt* only.

## 4. Analysis

In this section, we analyze the security and performance of our scheme.

### 4.1 Security

We claim that the proposed *TDS* algorithm is IND-CDA secure.

To prove the above claim, remind the game described in the section 2.3. In the game, $\mathfrak{A}$ can be considered as a pair of two polynomial-time algorithms, ($\mathfrak{A}_1$, $\mathfrak{A}_2$). $\mathfrak{A}_1$ generates two files {$F_0$, $F_1$} that hold the relation $Dic(F_0) = Dic(F_1)$, and selects an positive integer $j$ less than $N$. Then $\mathfrak{B}$ randomly chooses $i$ and $tk$ to run $C\text{-}Gen(F_i, \ell, N, tk)$= {$CK_1$, $CK_2$, …, $CK_N$}. Then a chunk $CK_j$ is given to $\mathcal{A}_2$ as a challenge. If the computed value $i' = \mathcal{A}_2(F_0, F_1, CK_j)$ is the same to $i$, then $\mathcal{A}$ wins the game. The advantage of $\mathcal{A}$ can be formalized as follows

$$\mathcal{ADV}_{\mathcal{A}} = |\ \Pr[\ i' = i \mid (F_0, F_1, j) \leftarrow \mathcal{A}_1, (i, tk) \leftarrow \mathcal{B},\\ \{CK_1, \ldots, CK_N\} \leftarrow C\text{-}Gen(F_i, \ell, N, tk), i' \leftarrow \mathcal{A}_2(F_0, F_1, CK_j)\ ] - 1/2\ |.$$

Here, we omitted considerations about extra *C-Gen* and *F-Ret* queries before or after the $\mathcal{B}$ generates the challenge. Note that each *C-Gen* or *F-Ret* algorithm runs using randomly selected *tk*. Therefore, we can easily deduce that there is no influence of existence of extra queries to the adversary's advantage from the property of the pseudo-random generators.

To prove the above claim, we first assume that there exists $\mathfrak{A} = (\mathfrak{A}_1, \mathfrak{A}_2)$ with non-negligible advantage. Then we will show that using $\mathfrak{A}$ one can easily distinguish a pair of two pseudorandom-strings which are generated from related inputs, from a pair of two pseudorandom-strings from independent inputs.

To formalize this proof, we need one more participant, $\mathfrak{C}$, who generates PRNG-challenges ($S_1$, $S_2$) such that $|S_1| = len_1$ and $|S_2| = len_2$. In the PRNG-challenges, an integer $j$ is given to $\mathfrak{C}$ and $\mathfrak{C}$ randomly chooses an integer $i_{\mathfrak{C}}$ from 0 or 1. If $i_{\mathfrak{C}} = 0$, then $\mathfrak{C}$ generates a pair ($prng(len_1, tk)$, $prng(len_2, tk \| j)$) as PRNG-challenge, where *tk* is random seed selected by $\mathfrak{C}$. Otherwise, $\mathfrak{C}$ generates a PRNG-chanllenges as ($prng(len_1, rs_1)$, $prng(len_2, rs_2)$) from randomly selected two random seeds $rs_1$ and $rs_2$. Note that from the property of pseudorandom generators it is hard for any polynomial-time algorithm to guess $\mathfrak{C}$'s choice from the PRNG-challenge with non-negligible advantage.

Finally, we state that if $\mathfrak{A}$'s advantage is non-negligible then $\mathfrak{B}$ can find correct answer for the PRNG-challenge with non-negligible advantage.

For the first step, $\mathfrak{B}$ starts the IND-CDA game with $\mathfrak{A}$. $\mathfrak{A}_1$ sends ($F_0$, $F_1$, $j$) to $\mathfrak{B}$. Then $\mathfrak{B}$ sends $j$ to $\mathfrak{C}$ and receives PRNG-challenge ($S_1$, $S_2$). Next, $\mathfrak{B}$ chooses $i$ and simulates $C\text{-}Gen(F_i, \ell, N, tk)$. However, in this simulation $\mathfrak{B}$ using $S_1$, $S_2$ as outputs of $prng(len_1, tk)$ and $prng(len_2, tk \| j)$ in the *C-Gen* function, respectively. $\mathfrak{B}$ sends $CK_j$ to the $\mathfrak{A}$ as a challenge of the IND-CDA game. If $\mathfrak{A}$'s guess $i'$ is the same to $i$, then $\mathfrak{B}$ answers ‘0’ to $\mathfrak{C}$. Otherwise $\mathfrak{B}$ answers ‘1’.

Therefore, the advantage of $\mathfrak{A}$ is negligible from the property of the pseudo-random number generators. It implies that the proposed algorithm is IND-CDA secure.

## 4.2 Performance

To measure the performance of the proposed technique, we count the number of additional operations compared with the naive approach where the target data to be stored is simply divided into $N$ chunks and stored to distributed storage servers without any additional data processing. As explained in Section 3.3, in our scheme, we need to perform the following additional operations:

- Generate $N$+1 random sequences.
- Generate the encoded file using random noises.
- Shatter the encoded file into shreds.
- Generate chunks by combining some shreds.
- Shuffle in each chuck.

The first operation requires $N$+1 pseudo-random function evaluations. Note that the size of random streams is much smaller than the size of the original file, and thus the cost for the first operation is not a burden. The encoded file is computed by inserting noise-values in the original file. It can be simply implemented by bitwise-shift and copy operations. The third and the fourth steps also can be easily implemented using iterative copy operations of small size bit streams. Note that, in the implementation, the last operation (shuffling in each chunk) is not considered since the operation was included for the simplicity of theoretic proof. Even if the operation is included, the last operation can be easily implementable as the other operations without large burden or it can be entrusted to storage servers after chunks are distributed.

We implement our scheme and compare its performance with AES encryption algorithm. For fair comparison, we use a well-known AES code in OpenSSL Ver.1.0.2. We use 10MB data and assume 5 servers. Specifically, the size of shreds is 32-bits, the ratio of random noise is pre-set as 10%. As shown in **Table 1**, the cost of encoding(shreding) and decoding are greatly reduced compared with AES encryption. Even though AES code is highly optimized differently from the proposed scheme, our scheme is efficient than AES. Two operations are about 3 times faster than AES encryption in the current implementation. With further optimization, it is expected that the performance of the proposed scheme will highly outperform the advanced encryption standard AES.

**Table 1.** Comparison

| | The proposed scheme | | AES (CBC mode) |
|---|---|---|---|
| Operation | Shreding | Decoding | Encryption |
| Time | 0.026499 sec | 0.028725 sec | 0.088902 sec |

In **Table 2**, we test the effect of the size of shreds. Intuitively, we can expect that the cost of encoding will decrease since the number of unit operations is decreasing for large shreds. The intuitive idea regarding the cost of encoding can be verified in **Table 2**. As seen in the table, the cost of encoding is decreasing for larger shreds, but the effect of changing the size of shreds is not crucial for the total efficiency.

**Table 2.** Encoding time for various shred sizes

| Shred size | 16 bits | 32 bits | 64 bits | 128 bits | 256 bits |
|---|---|---|---|---|---|
| Time | 0.028518 sec | 0.026419 sec | 0.025739 sec | 0.024886 sec | 0.024820 sec |

One of performance issue is the number of distributed storages. In **Table 3**, we tested the cost of encoding for various number of servers. As a result of simulation, we can see that the cost does not highly influenced by the number of server. Note that the number of servers also influences on the security of the proposed system since an adversary may try to obtain entire data chunks by corrupt all storage servers. To prevent the adversary from gain all chunks, we need to adopt at least $n$ storage servers where $n$ is a number which is great than the maximum number of corrupted servers. Fortunately, as seen in **Table 3**, the cost of encoding does not highly increase according to the number of servers, which means that we can easily increase the number of storage servers so that the adversary cannot corrupt all storage servers.

**Table 3.** Encoding time for various number of servers

| # of servers | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| Time | 0.026948 sec | 0.026813 sec | 0.026575 sec | 0.027007 sec | 0.027303 sec | 0.027794 sec | 0.027769 sec | 0.028284 sec | 0.027608 sec |

## 5. Variance

Since we use random noises to randomize positions of shreds, encoded file cannot be de-duplicated. In this section, we will give a modification of our scheme, which can support the de-duplication functionality. Before giving concrete description of de-duplicatable trapdoor digital shredding, we give brief review of existing techniques for the de-duplication of randomized file. Then, we will give a de-duplicatable trapdoor digital shredding.

### 5.1 Review of Existing De-duplication Technique

De-duplication is a tool for eliminating repeated copies to save the cost of storage. Due to the importance of de-duplication, many researchers tried to give secure and efficient de-duplication technique [12, 8, 9]. However, until the convergence encryption has been proposed in [7], we cannot de-duplicate encrypted data. In 2013, Bellare *et al*. proposed a class of de-duplicatable encryption techniques so-called message-locked encryption whose security is verified under formal security proof [4]. In [4], four schemes have been proposed with provable security. Note that existing de-duplicatable encryption techniques can be seen as variations of the basic convergent encryption in [7]. After the theoretic progress due to Bellare *et al*., many improved techniques have been made in terms of the security and the efficiency [5].

In general, encrypted files are not easy to de-duplicate since different ciphertexts are computed for the same file if different keys are used for encrypting the file. For security, each user uses his own secret key, and thus the ciphertext for the user is different the ciphertext generated by another user. To support the de-duplication for encrypted files, the main idea of de-duplicatable encryption scheme is to use the same key for the same data by deriving the encryption key from the data itself. Since clients can obtain the same key for the same data, the ciphertext is also the same.

To support the de-duplication functionality in our scheme, we will adopt the basic idea of de-duplicatable encryption. In the following section, we describe our strategy for giving the functionality with concrete description.

### 5.2 Trapdoor Digital Shredding with De-duplication

Though there are many duplicated explanation, we will give the description in detail to help the reader understanding our idea. As in the basic scheme, we let *Clt* be the client who wants to store a file $F$ to $N$ servers $\{Srv_1, Srv_2, \ldots, Srv_N\}$. The trapdoor digital shredding system supporting de-duplication works as follows.

- **Upload.** To upload the file $F$, the client *Clt* generates an identifier $id_F$ for the file and runs the trapdoor digital shredding function *TDS*() to generates chunks to transmit each storage server. The trapdoor digital shredding function runs mainly in two steps;

  1. compute $tk_F$ from $F$ by running the pseudo random generator by using the file as an input;
  2. generate a set of random information which can be computed as following:
     $$n_1 \| j_1 \| n_2 \| j_2 \| \cdots \| n_k \| j_k = prng(len_1, tk_F);$$
  3. generate $EF$ using the random noise as in the basic scheme;
  4. divide $EF$ into $tN$ shreds;
  5. construct $N$ subsets as $S_j = \{P_i \mid i = j \bmod n\}$;
  6. compute $p_1 \| p_2 \| \cdots \| p_t = prng(len_2, tk_F \| j)$ and shuffle each subset as in the basic scheme;
  7. set $CK_j$ as concatenation of all shreds in $S_j$; and
  8. upload $CK_j$ to the $j$-th storage server.

- **Download.** As in the basic scheme, to retrieve the file $F$, the client downloads $CK_i$ to $Srv_i$ for all $i = 1, \ldots, N$, and re-constructs the file using the private trapdoor key $tk$.

As seen in this section, the only difference between the basic scheme and the de-duplicable scheme is the seed value $tk_F$ used for evaluating the random noises. Differently from the basic scheme, in the variation, we use the file to be store in a distributed way is used as the input for the random generation function. Then we can expect that a file is encoded to the same chunks, which permits the de-duplication of the same copies.

Since the only difference is the randomness of $tk_F$ which is influenced by the knowledge regarding the file to be stored, we can say that the security of the variance is identical with the security of the basic scheme. The additional cost for supporting the de-duplication is to compute the random value from the file instead of simply generating it.

## 6. Conclusion

Until now, the only way to secure stored information is to encrypt the data using well-known cryptographic algorithms. However, in great part of storage services, service providers are not use the solution due to the cost of encryption. In this paper, we proposed a new and novel technique for securing data without using encryption scheme. We also give formal security analysis for our scheme.

## References

[1] United States National Institute of Standards and Technology (NIST), Advanced encryption standard (AES), Federal Information Processing Standards Publication 197, 2012.

[2] A. Bessani, M. Correia, B. Quaresma, F. Andre and and P. Sousa, "DEPSKY: Dependable and Secure Storage in a Cloud-of-Clouds," *ACM Transactions on Storage*, vol. 9, no. 4, Article 12, 2013. Article (CrossRef Link)

[3] G. Bertoni, L. Breveglieri, P. Fragneto, M. Macchetti and S. Marchesin, "Efficient Software Implementation of AES on 32-Bit Platforms," in *Proc. of CHES 2002*, LNCS vol. 2523, pp. 159-171, 2003. Article (CrossRef Link)

[4] M. Bellare, S. Keelveedhi and T. Ristenpart, "Message-Locked Encryption and Secure Deduplication," in *Proc. of EUROCRYPT 2013*, LNCS vol. 7881, pp. 296-312, 2013. Article (CrossRef Link)

[5] M. Bellare, S. Keelveedhi and T. Ristenpart, "DupLESS: Server-Aided Encryption for Deduplicated Storage," in *Proc. of the 22nd USENIX conference on Security*, pp. 179-194, 2013. Article (CrossRef Link)

[6] R. Bitar and S. El Rouayheb, "Securing Data against Limited-Knowledge Adversaries in Distributed Storage Systems," in *Proc. of 2015 IEEE International Symposium on Information Theory (ISIT)*, pp. 2847-2851, 2015. Article (CrossRef Link)

[7] J. R. Douceur, A. Adya, W.J. Bolosky, P. Simon and M. Theimer, "Reclaiming Space from Duplicate Files in a Serverless Distributed File System," in *Proc. of Distributed Computing Systems*, pp. 617-624, 2002. Article (CrossRef Link)

[8] S. Halevi, D. Harnik, B. Pinkas and A. Shulman-Peleg, "Proofs of Ownership in Remote Storage Systems," in *Proc. of 18th ACM. Conf. Computer and Communications Security, ACM Press*, pp. 491-500, 2011. Article (CrossRef Link)

[9] D. Harnik, B. Pinkas and A. Shulman-Peleg, "Side channels in cloud services : Deduplication in cloud storage," *IEEE Security&Privacy*, vol. 8, no. 6, pp. 40 - 47, 2010. Article (CrossRef Link)

[10] K. K. Mar, Z. Hu, C. Y. Law and M. Wang, "Secure Cloud Distributed File System," in *Proc. of The 11th International Conference for Internet Technology and Secured Transactions (ICITST-2016)*, pp. 176-181, 2016. Article (CrossRef Link)

[11] V. Kumari and S. Chakravarthy, "Efficient implementation of the AES algorithm for security applications," in *Proc. of System-on-Chip Conference*, pp. 206-210, 2016. Article (CrossRef Link)

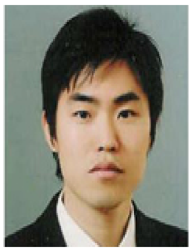
[12] M. W. Storer, K, Greenan, D. D. E. Long and E. L. Miller, "Secure data deduplication," in *Proc. of 4th ACM International Workshop on Storage Security and Survivability*, pp. 1-10, 2008. Article (CrossRef Link)

[13] B. N. Van, S.-H. Lee and K.-R. Kwon, "Selective Encryption Algorithm Using Hybrid Transform for GIS Vector Map," *Journal of Information Processing Systems*, vol.13, no.1, pp. 68-82, 2017. Article (CrossRef Link)

[14] V. Kumari and S. Chakravarthy, "Cooperative privacy game: a novel strategy for preserving privacy in data publishing," *Human-centric Computing and Information Sciences 2016*, 2016. Article (CrossRef Link)

[15] T. Zhu, X. Zou and J. Pan, "Query with SUM Aggregate Function on Encrypted Floating-Point Numbers in Cloud," *Journal of Information Processing Systems*, vol.13, no.3, pp. 573-589, 2017. Article (CrossRef Link)

**Dr. Taek-Young Youn** received his BS, MS, and Ph.D from Korea University in 2003, 2005, and 2009, respectively. He is currently a senior researcher at Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea. From 2016, he serve as an associate professor in University of Science and Technology (UST), Daejeon, Korea. His research interests include cryptography, information security, authentication, data privacy, and security issues in various communications.



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