Received: 26 December 2018 | Revised: 4 April 2019 | Accepted: 29 April 2019
DOI: 10.1002/cpe.5350

WILEY

SPECIAL ISSUE PAPER

Sed-Dedup: An efficient secure deduplication system with data modifications

Wenlong Tian[1] | **Ruixuan Li**[1] | **Cheng-Zhong Xu**[2,3] | **Zhiyong Xu**[4,5]

[1]School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China
[2]Electrical and Computer Engineering, Wayne State University, Detroit, Michigan
[3]State Key Laboratory of IoTSC and Department of Computer Science, University of Macau, Macau SAR, China
[4]Math and Computer Science Department, Suffolk University, Boston, Massachusetts
[5]Shenzhen Institute of Advanced Technology, Chinese Academy of Science, Shenzhen, China

Correspondence
Ruixuan Li, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China.
Email: rxli@hust.edu.cn

Present Address
Ruixuan Li, Huazhong University of Science and Technology, Wuhan 430074, China

Funding information
National Key Research and Development Program of China, Grant/Award Number: 2016YFB0800402 and 2016QY01W0202; National Natural Science Foundation of China, Grant/Award Number: U1836204, 61572221, 61433006, U1401258, and 61502185; Major Projects of the National Social Science Foundation, Grant/Award Number: 16ZDA092; Guangxi High level innovation Team in Higher Education Institutions--Innovation Team of ASEAN Digital Cloud Big Data Security and Mining Technology

Summary

The amount of outsourced data grows rapidly. In recent years, cloud service providers integrate data deduplication systems with convergent encryption (CE) methods, in which a file encryption key is determined by its own content instead of the secret of a specific user, to save the storage cost and ensure the security of outsourced data. However, present secure deduplication systems failed to deal with data modifications efficiently. We observe that when a client makes small changes on an existing file, the current chunking algorithms cannot effectively detect the similarities and always create chunks with largely overlapped contents. It reduces data deduplication ratios and results in unnecessary overhead. In this paper, we propose Sed-Dedup, an efficient secure delta encoding deduplication system to address this problem. In Sed-Dedup, we introduce a novel delta encoding approach to store modified contents in delta files and leave the original files intact. Two schemes with different encoding policies are designed. Both of them can solve the issue and improve the secure deduplication performance. To evaluate the performance, we implement a prototype and conduct extensive experiments based on synthetic and real-world datasets. Our experimental results show that Sed-Dedup is superior to the state-of-the-art secure deduplication systems.

KEYWORDS

chunking algorithms, convergent encryption, data modifications, delta encoding, secure deduplication

1 | INTRODUCTION

With the rapid growth in cloud technology, the cloud storage service attracts great interest. It encourages people to migrate their data onto remote servers by offering high data reliability and availability with low maintenance cost. Cloud service providers (SP) such as Google Drive,[1] Microsoft OneDrive[2] and Baidu Cloud[3] are getting increasing popular among enterprizes and individual customers. However, the ever rising amount of data generates great data storage, network communication and maintenance burdens. Providing efficient redundant data elimination mechanisms becomes a critical challenge.[4,5] Various techniques have been proposed. Data deduplication is one of the most widely used techniques. It achieves great successes and has been adopted in numerous places including Dropbox,[6] Mozy,[7] etc.

Data deduplication methods can be categorized in various manners. The most common strategy divides it into file-level and chunk-level deduplications based on data partitioning granularity.[8] However, in file-level, the amount of data duplications it can remove is limited. In chunk-level deduplication, files are divided into chunks, and data replication checks are performed at the chunk-level. Although the overhead is higher, it can achieve better data deduplication ratio. Thus, it is more popular than the first approach in the state-of-the-art deduplication systems.

However, as the data volume stored on the cloud increases exponentially, consumers are becoming increasing worried about the confidentiality of their data. How to ensure the security and privacy of outsourced data draws high interest from both industry and academia.[9,10] Traditionally, a symmetric encryption algorithm such as DES[11] or AES[12] can be chosen for data protection. Although it can ensure the data security, it poses great challenges on data deduplication. For instance, if two users share some common files or chunks, since they use different keys for encryption, the generated ciphertexts are not the same. Thus, the service provider (SP) cannot discover duplications.

Previous works use Convergent Encryption (CE)[13] or Message-Locked Encryption[14] to formalize secure deduplication. In CE mechanism, the key used to encrypt a file or chunk is not bind to a particular owner. It is determined by the data itself and is generated with a collision-free hash function (such as SHA-1[15]). Thus, the ciphertext of the same plaintext data shared by multiple users remains the same. SP can examine the redundancy on encrypted data, and execute deduplication operations if necessary. A lot of systems using CE have been implemented.[16-18] However, such an approach results in a large number of encryption keys, and the number of these keys expands rapidly as the data volume amplifies. The encryption key management becomes a headache and greatly reduces the system overall performance. To solve this problem, SecDep[18] introduces User-Aware Convergent Encryption and Multi-Level Key management strategy. Liu et al[19] propose a scheme that supports client-side encryption without requiring any additional independent servers. Tang et al[20] aim to do secure deduplication on ciphertext by a new ciphertext deduplication techniques. Nevertheless, none of them addresses the data modification problem effectively.

For normal users, beyond the operations of uploading the data to the cloud and retrieving their outsourced data from the cloud, making modifications on their existing data are also very common. Meister and Brinkmann[21] analyze how small changes affect the deduplication ratio for different file types on a microscopic level for chunking approaches. For example, office workers often synchronize multiple version of their daily documents through the online backup service. Databases administrators also dump database tables on a daily basis. GIT version control systems[22] and Linux Kernel archives[23] are other examples. Current secure data duplication solutions cannot solve this issue very well. It can add large amount of unnecessary overheads such as extra chunk generations, redundant encryptions, excessive data transmissions and additional encryption key generations, etc. For instance, in SecDep, when a user uploads a file which is a new version of an existing outsourced file, the key(s) used to encrypt the new version are not the same for the old version with the CE mechanism since the encryption keys are generated based on the content (file or chunks). Thus, the ciphertext is completely unrelated even if the new version is only slightly different from the old one. It results in the following issues. First, redundant data between consecutive file versions cannot be effectively deduplicated, extra storage spaces are needed to store vastly overlapped data files/chunks. Second, the amount of network transmission also increases since more data need to be uploaded. Consequently, it adds extra processing time as well. Third, more encryption keys are generated. It increases key management overheads. As time evolves, more data modifications are made and these issues become more severe.

To overcome the above problems, we propose Sed-Dedup, an efficient secure deduplication mechanism. It introduces a delta encoding protocol[24-26] to avoid the redundant storage overhead, reduce the network traffic and relieve the excessive key generation problem. Instead of uploading complete new modified files, in Sed-Dedup, we only create delta files to keep the modified portions. Two schemes with different policies are designed. In Sed-Dedup-I, we record the difference between an original file and a modified file in a delta file. It uses the corresponding delta file's file-level key as the modified file's file-level key. In Sed-Dedup-II, each delta file only records the modified portion of a file with its nearest predecessor version. Each modified file's file-level key is the same as the corresponding delta file's file-level key. To the best of our knowledge, Sed-Dedup is the first system designed to address data modification problem in secure deduplication systems. Overall, our contribution are as follows:

- First, we analyze the data modification problem in current data deduplication systems, and find that it has negative impacts on duplication detection efficiency in content defined chunking algorithms. It can generate many new chunks with large overlaps on original chunks.
- Second, we design a novel delta encoding protocol, it tracks modified portions among multiple version files. With this approach, we decrease the key management overhead and also avoid the redundant storage overhead caused by data modifications. The data deduplication performance is greatly improved, especially for systems with frequent changes.
- Third, we introduce two schemes with different strategies to record modified portions in delta files. We also develop new retrieving strategy to save the file restore time. Our designs greatly reduces network traffic, decreases processing time and lessens key management overhead.
- Finally, we design and implement a prototype of Sed-Dedup to evaluate its performance. Experimental results show that, based on synthetic and real world datasets, it has superior performance compared with the state-of-the-art SecDep scheme.

The rest of the paper is organized as follows. In Section 2, we formalize the issues in current secure deduplications. In Section 3, we present Sed-Dedup system design. In Section 4, we discuss the operational implementation. In Section 5, we first introduce the experimental simulation configurations. Then, we describe Sed-Dedup performance evaluation compared with SecDep scheme. In Section 6, we analyze the security issue in Sed-Dedup. In Section 7, we introduce related works. Finally, in Section 8, we conclude the paper and give the future work.

TABLE 1 Commits per author per quarter on Linux kernel

Author	Q2 2016	Q3 2016	Q4 2016	Total
H Hartley	135	147	284	566
Geert	166	143	245	554
Mauro	207	56	211	474
Arnd	75	41	331	447
Mateusz	87	340	16	443
Linux	73	54	265	392
Thomas	133	179	48	360
Others	15 826	15 428	19 992	51 246
Total (3542)	16 702	16 378	21 392	54 482

2 | PROBLEM DESCRIPTION

Sed-Dedup aims to design new strategies to solve data storage and communication burdens as well as encryption key management overhead associated with data modifications in secure deduplication systems. In this section, we formalize the problem.

2.1 | Data modification and multiple versions

Small changes made between consecutive file versions is not unusual in real life. For example, in GIT version control system (https://github.com/), a project can easily have hundreds of commits submitted by a group of developers. Each commit represents a new version which introduces modifications (insert, update and delete) on a few existing files in the previous commit and/or add some new files. Many other applications also have the same feature. For example, the database dump file backups.

In this paper, we choose Linux kernel source code archives (http://www.kernel.org) and databases dump files to analyze data modification issues. Table 1 shows that individual authors commit many modifications per quarter and the number of people involved is also high. There are 3542 authors made a total of 54 482 commits. We examined the logs which record the difference between consecutive commits and found that most modifications only add/modify/delete a few lines in a limited number of files.

2.2 | Problem statement

The state-of-the-art secure deduplication systems such as SecDep cannot deal with the data modification problem. To describe this problem, we denote that when a file is first uploaded by a user onto the cloud, it is called an original file F_O. A file generated by making some modifications on F_O is called a modified file (F_{M1}). Then, another file generated by modifying F_{M1} is called F_{M2}, and so on. In SecDep, the system determines if a file (either a F_O or a F_{MK}) has already been uploaded to the cloud by checking the file tag. If the corresponding file tag does not exist, the file is split to chunks. The system checks if any of the chunks has already been uploaded using the chunk tags. Finally, only the chunks not stored on the cloud are encrypted and uploaded. Clearly, in most cases, F_O and F_Ms have to be split to chunks. The effectiveness of detecting redundant contents in chunks is highly depended on the amount and the positions of modifications. Although different chunking algorithms have distinct features and some can discover more common chunks than others, none of them can successfully detect most common contents.

We conduct an experiment to evaluate the efficiency of various chunking algorithms on data modifications. The result is shown in Table 2. Three commonly used chunking algorithms, including fixed-sized chunking, content defined chunking (CDC),[27] and Two threshold and Two Divisors chunking algorithm (TTTD)[28] are chosen. F_O is a randomly generated file which has the size of 1M. F_M is created by randomly updating F_O, the amount of modification is 0.01 MB. The average chunk size is 2 kB.

TABLE 2 Chunking algorithms comparison

Algorithm	A[a]	B[b]	C[c]	D[d]	E[e]
Fixed-size	512	518	1030	0	518
CDC	481	490	680	291	199
TTTD	467	473	640	300	173

[a]The total number of data chunks in F_O.

[b]The total number of data chunks in F_M.

[c]The total number of data chunks to be uploaded.

[d]The number of duplicate chunks detected.

[e]The number of new data chunks to be uploaded for F_M.

As we can observe from the result, fixed-sized chunking cannot detect any overlaps between F_O and F_M, and has to create 518 new chunks. CDC[27] and TTTD[28] work better, but still generate 199 and 173 extra chunks, respectively.

Clearly, the cloud storage is not efficiently utilized since it contains many similar chunks. The data deduplication performance is expected to be worse as more modifications are added.

Recently, some new chunking algorithms without any help from server are proposed.[4,29-35] However, most of them are dedicated to improve the throughout of chunking process. In terms of the dedup ratio metric, these new algorithms have similar performance as TTTD. In this paper, we focus on the reduced deduplication ratio problem caused by frequent small data modifications. Thus, we decide to choose the representative TTTD and sliding-window CDC and fixed-size chunking algorithm to show the impact.

3 | SED-DEDUP SYSTEM ARCHITECTURE

To solve the above problems, Sed-Dedup introduce a novel delta encoding protocol and two schemes with different encoding policies. In this section, we introduce Sed-Dedup architecture in detail. To simplify the discussion, we denote entities in a secure deduplication system in Table 3.

3.1 | System overview

We assume clients use the cloud storage service as a real-time backup system and make modifications regularly. In Sed-Dedup, we study the following two major operations: upload and restore.

When a user wants to upload a file, we consider two scenarios. If it is the first time to be uploaded by any user, we denote this file as an original file F_O. In this scenario, Sed-Dedup takes the same process as SecDep to handle the file backup operation. First, the K_O is generated with CE. Next, the system applies the tag algorithm to produce T_O and sends it to the SP. SP searches its internal data store to check whether T_O is already uploaded by another user or not. If yes, the process finishes after informing the current user a successful backup signal. If not, F_O has to be split into chunks. The chunk-level keys are generated with CE again and the chunk tags are created with the same tag algorithm as well. The chunk tag information is sent to the SP to execute the duplication check. Only the chunks not stored on the SP are uploaded.

In the second scenario, the file to be uploaded is a new version of an original file, F_M. Unlike the previous solutions which treat it as another F_O, Sed-Dedup introduces a novel delta encoding protocol by only recording the modification part of F_M to avoid the chunking problem we mentioned in Section 2. Then, the delta file, which records only the difference, is treated as a new file instead of the whole F_M and is stored on the SP. The delta file's file-level key is treated as F_M's file level key. Note that choosing different strategies in delta encoding process will affect the performance. We propose two schemes and the details will be discussed later.

When a user wants to retrieve a file from the SP, we also consider two scenarios. If the user asks for a F_O, Sed-Dedup uses the same strategy as SecDep to download the file. If the requested file is a F_M, the retrieving strategy depends on which Sed-Dedup scheme is chosen. The fundamental principle is to download only the ciphertext of the delta-encoded contents. In Sed-Dedup, we assume a user treats the SP as a replication service for its local files in most cases. Therefore, when a user uploads a file onto the SP, the user's local file system always stores a copy. In this case, a user might only need to download the corresponding delta file(s) from SP to restore the F_M (May created by another user). Certainly, the original file may be deleted after uploaded onto the SP. This is the worst scenario, the user needs to download F_O as well as the corresponding delta file(s) to recover F_M.

TABLE 3 Notations

Notation	Description
F_O	An original file
K_O	The file-level key of the original file
T_O	The file tag of the original file
P_O	The plaintext of an original file
F_M	A modified file
F_{Mi}	The i-th modified version of the file F_O
T_M	The tag of a modified file
P_M	the plaintext data of a modified file
DKS	Distributed key servers
SP	The storage server
CE	Convergent Encryption
SSSS	Shamir Secret Sharing Scheme

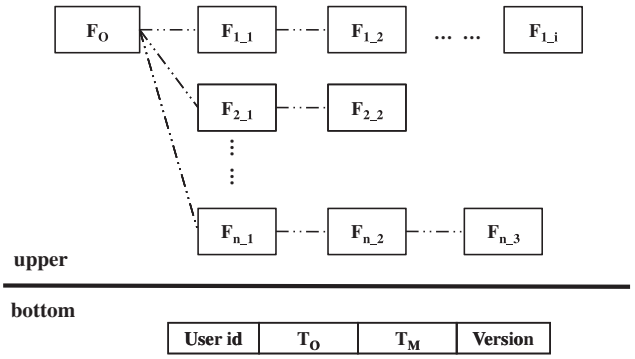

FIGURE 1 A sample inheritance map

3.2 | Delta encoding protocol

In Sed-Dedup, we associate a F_O with its consecutive F_Ms for more effective treatment. We use an inheritance map, as shown in Figure 1, to maintain the relationships between an F_O and multiple successive F_Ms. When a user sends a request to backup a file which is a F_M of an existing F_O, delta encoding protocol is executed. Here, we assume F_O or a previous $F_{M'}$ (the comparison data) is stored on the user's local space (the user downloads it if it does not exist). The user communicates with the SP and verifies if F_M has already been stored in the cloud using the file-level tag of F_M. If not, the delta encoding process starts.

A delta file is generated on the user's local machine by examining the difference between the comparison data and F_M, and it is stored in the plaintext format first. Delta files are stored in ciphertext on the SP to ensure the data security. Thus, after a plaintext delta file is created, we apply CE to generate the file-level key of F_M. If the modification in a delta file is larger than a chunk, multiple encrypted chunks are created. Finally, those chunks are uploaded onto the SP. Clearly, with the delta encoding protocol, Sed-Dedup minimizes the redundant chunks. Such an approach is very efficient for files with multiple consecutive modifications.

3.3 | Sed-Dedup-I

We design two schemes for Sed-Dedup. In Sed-Dedup-I, each of the consecutive versions generates a delta file by comparing with the F_O directly. When a user wants to upload a modified F_M onto the SP, T_M is calculated first. Then, the server checks whether this version has be uploaded or not; if not, a delta file is created by recording the difference between the F_M and F_O (already stored on the SP). Next, a file-level key is generated with CE based on the content of the delta file. The user only need to upload the ciphertext of the delta file instead of the ciphertext of entire F_M.

Sed-Dedup-I can change a file F_{ij}'s store mode into the original mode when a user wants to share it with other users. Here, i denotes the modified file id, j denotes the version number. Then, $F_{i(j+1)}$ and later versions use F_{ij} as a an original file during the delta encoding process. In this case, the overhead of sharing modified version data with others is very high. When many users share all their versions, it essentially becomes SecDep. This issue is relieved in Sed-Dedup-II. The detail is presented in Section 3.4.

Another problem of Sed-Dedup-I is that as the number of versions increases, the difference between the newest F_M and F_O increases, so does the size of the delta file. This problem can be relieved as the delta files for multiple versions can share some common data and deduplicated. However, it is still higher than Sed-Dedup-II.

An advantage of Sed-Dedup-I is that, when a user wants to restore a F_M, the user only need to download one delta file for the modified portion.

3.4 | Sed-Dedup-II

In Sed-Dedup-II, a delta file is generated by comparing $F_{i(j+1)}$ with F_{ij}. Here, $F_{i(j+1)}$ is compared directly with its immediate predecessor F_{ij}. Thus, each delta file only records the modifications between two consecutive versions. As Sed-Dedup-I, the client also uses the standard CE and tag generation algorithms to produce the file-level key and the file tag for $F_{i(j+1)}$.

Clearly, the main difference between these two schemes is which data is chosen as comparison data in delta encoding process and the restore process. Sed-Dedup-II chooses the previous version as comparison data instead of the original data. In the restore process, it may need to fetch several delta files to reach the target modified version, while Sed-Dedup-I only need to fetch one delta file in most scenarios. Both of them also need to download F_O in the worst case.

For data sharing, Sed-Dedup-II only changes F_{ij}'s store mode into the original mode, when a user wants to share it with other users. However, there is no need to change the subsequent version's delta file since it is the comparing result between the modified version and its previous version. So the overhead of sharing modified version is much lower.

3.5 | Data encryption scheme

In Sed-Dedup, for a F_O, three kinds of keys exist, including the file-level key, the chunk-level keys and the share keys. They have different functionalities. Both the file-level key and the chunk-level keys are traditional CE keys, respectively associated with the corresponding file and chunk content. They should be kept secret during encryption/decryption processes on the SP. Thus, chunk-level keys are encrypted with the corresponding file-level key and stored on the SP. In SecDep, to prevent single point of failure problem, share keys are created for each file-level key using the SSSS algorithm, and those keys are distributed over multiple key servers in DKS. In Sed-Dedup, we use the same approach.

For a modified file F_M, we create the file-level key and chunk-level keys on each delta file based on the same algorithm, and then use this file-level key as the F_M's to encrypt chunk level keys. Finally, those encrypted keys are stored on the SP. With this approach, we decrease the chunk-level keys for each F_M, and greatly reduce the burden on key management and improve deduplication performance.

4 | IMPLEMENTATION DETAILS

Sed-Dedup consists of the following entities including the storage servers (SP), the distributed key servers (DKS) and the clients. Similar to SecDep, DupLESS, and Dekey, standard encryption and signature algorithms such as SHA-256, MD5, and AES-256 are used. In Sed-Dedup, we implement chunking algorithms, file-level/chunk-level key generation, tag generation, and delta encoding/decoding algorithms. We also implement the functionalities on the SP and DKS.

4.1 | Chunking algorithm

Chunking operation is very important in data deduplication systems. There are many chunking algorithms have been developed such as Fixed-size Chunking, Content Defined Chunking,[27] and Two Thresholds and Two Divisors Chunking,[28] etc. In Sed-Dedup, the default chunking algorithm is TTTD. Four parameters are used, namely, maxT, minT, mainD, and secondD. maxT and minT define the maximum and minimum chunk sizes and are used to avoid generating too large or too small chunks. mainD is an integer divisor which plays the same role as in Basic Sliding Window algorithm (BSW).[36] In general, the value of secondD is half of mainD.

TTTD in Sed-Dedup system works as follows. First, it tries to create the fixed-size window W. It shifts one byte each time from the head to the end. When the size of the window reaches the minT, it starts to determine the backup breakpoint and real breakpoint with secondD and mainD. The process includes two formulation tests (h mod secondD) == (secondD-1) and (h mod mainD) == (mainD-1), where h is a hash value for the current content in W. The hash value h is generated by Rabin Fingerprinting (RF) algorithm[27] with the window content, h = RF(W). If the position fits the second formulation test, it is treated as a backup breakpoint. If it also passes the first formulation test, the backup breakpoint is chosen as a real breakpoint. When the algorithm cannot find a real breakpoint after reaching the threshold, maxT, maxT is chosen to generate the breakpoint in W. In Sed-Dedup, we choose the parameters in TTTD according to the experimental results in Kave and Khuern.[28]

4.2 | Delta encoding and decoding algorithm

To achieve better deduplication performance, Sed-Dedup only records the difference of a F_O/F_M and a consecutive F_M in delta files. Delta encoding algorithm with the VCDIFF (https://tools.ietf.org/html/rfc3284) format is used. It is a general, efficient, and portable data format suitable for encoding compressed and/or differencing data. VCDIFF uses three types of instructions, namely, ADD, COPY, and RUN. Detailed explanation of these instructions are in Table 4. We describe VCDIFF with an example in Figure 2.

As Figure 2 shows, T is a target data and S is a source data. A delta file is generated to record the difference between S and T. When a client tries to restore T, Table 5 shows the reconstruction process. Initially, the temporary target data is empty. The system executes the instructions one by one in the delta file. With the COPY 4, 0 instruction, the first 4 characters in S is copied to the temporary target. Next, the ADD 4, wyxz instruction results in the characters wxyz are added to the end of the temporary target. The operation continues, until eventually all the instructions in the delta file are executed. Finally, we successfully reconstruct the target data T.

TABLE 4 Instruction in VCDIFF

Instruction		Explanation
ADD	x, y	x is a size, y is a sequence of x bytes to be copied
COPY	x, y	x is a size, y is the address in superstring U, the substring of U will be copied, where U: S[0]S[1].....S[s-1]T[0]T[1].....T[t-1]
RUN	x, y	x is a size, y is a sequence that will be repeated x times

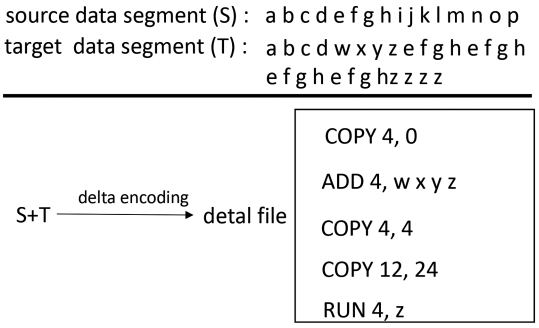

FIGURE 2 Sample delta file

TABLE 5 Reconstruction process

Delta Instruction	Restored Target Chunk for Each Instruction
COPY 4,0	abcd
ADD 4, wxyz	abcdwxyz
COPY 4, 4	abcdwyxzefgh
COPY 12, 24	abcdwyxzefghefghefghefgh
RUN 4, z	abcdwyxzefghefghefghefghzzzz

5 | PERFORMANCE EVALUATION

To evaluate Sed-Dedup performance, we conduct extensive simulation experiments to compare it with the state-of-the-art secure deduplication system, SecDep. In this section, we first introduce experimental configurations, and then we present evaluation results compared with SecDep.

5.1 | Experimental setup

All the experiments are conducted on a distribution platform. Each machine is equipped with a Intel(R) Core(TM) i7-4702MQ @2.20 GHz 8 core CPU, 8 GB RAM, a 1 TB 5400 rpm hard disk and is connected in a 100 Mbps network. The OS installed is Ubuntu 15.10 LTS 64-bit System.

Due to the lack of the SecDep source code, we implement it by ourselves. We also implement Sed-Dedup. Both are written in Java. We use SecDep as the baseline. We use the following metrics, backup size, backup time, encryption key storage space overhead, delta encoding time for evaluation. Note that our evaluation platform is not a commercial quality system but rather a research prototype. Hence, our evaluation results should be interpreted as an approximate and comparative assessment. Liu et al[19] designed a method to relieve the key management overhead with client-side encryption, and does not require the assistance from any independent servers. However, it only works for file-level encryption. When applied to block-level encryption, it works poorly. In Sed-Dedup, we focus on th performance on the block-level encryption. Thus, we do not compare Sed-Dedup with the work of Liu et al.[19]

In our experiments, we use three types of datasets. The first type consists of a group of artificial files created with randomly generated content. Each file has a version number and consecutive files have randomly modifications on a previous version. The second dataset contains multiple versions of Linux kernel source code from (http://www.kernel.org). As we mentioned in Section 2, these versions contain a large number of small updates which are good representations of the data modification problem. The third dataset contains multiple versions of mysql dump files, which also has various amount of consecutive modifications.

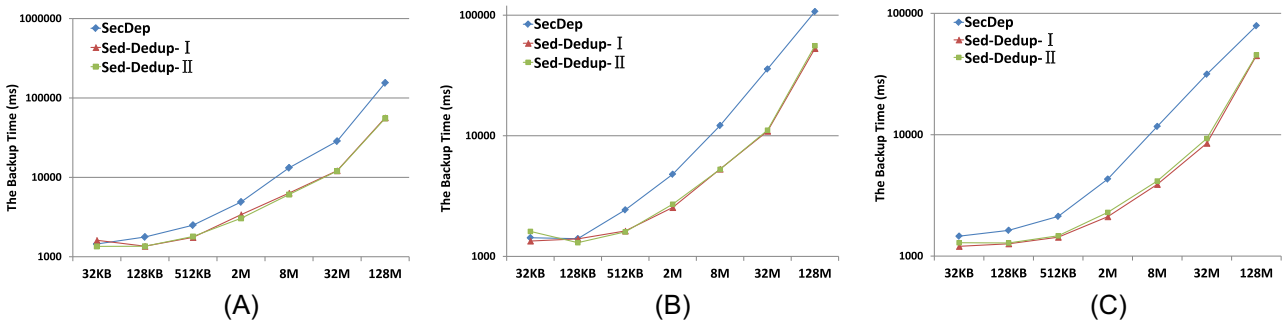

FIGURE 3 Single file random modification time comparison result under two-versions scenario. A, Average chunk size: 2 kB; B, Average chunk size: 4 kB; C, Average chunk size: 8 kB

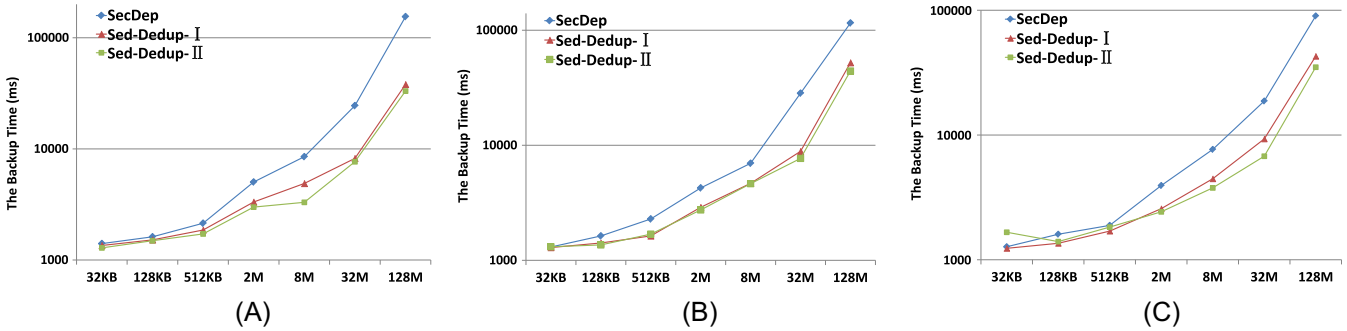

FIGURE 4 Single file random modification time comparison result under three-versions scenario. A, Average chunk size: 2 kB; B, Average Chunk size: 4 kB; C, Average Chunk size: 8 kB

5.2 | Backup time comparison for single file modification

First, we check the backup time performance with one and two consecutive modifications in the first dataset. In this set of experiments, the sizes of the original file vary from 32 kB to 128 MB. Those files are created with random content. Each time, the amount of modification is set as 3% of the original file size. To evaluate the impacts of average chunk sizes, we conduct our experiments with 3 different average chunk sizes: 2 kB, 4 kB, and 8 kB. Figure 3 shows the backup result of one original file and one modified version file (Two-Versions Scenario). Figure 4 is the backup result with two modified version files (Three-Versions Scenario).

As we can observe from Figure 3A, in all scenarios, Sed-Dedup takes much shorter time. Furthermore, Sed-Dedup-I and Sed-Dedup-II have the similar performance dealing under Two-Versions scenario. It also shows that, as the file size increases, the backup time difference becomes larger. When the F_O size is 128 MB, Sed-Dedup only takes 56 390 ms to backup the difference while SecDep needs 154 754 ms to finish the operation. Sed-Dedup only takes 36.44% of time SecDep consumes. On average, Sed-Dedup takes 67.11% of time SecDep needs to backup. Apparently, by using delta encoding protocol, the amount of data transmission is also greatly reduced, so does the total operation time. The same trend holds as we can observe from Figures 3B and 3C. With the average chunk sizes of 4 kB and 8 kB, Sed-Dedup outperforms SecDep for almost all the scenarios.

Since the increasing number of versions can make delta files become large quickly in Sed-Dedup-I, there is still some space to improve by using Sed-Dedup-II. To evaluate the difference, we examine the backup performance on Three-Versions scenario. The results are shown from Figure 4A to 4C. On average, it takes much less time in Sed-Dedup-II. We also conduct experiments on even more number of consecutive modified versions and the results show the same trend.

5.3 | Backup time analysis

To further understand the benefits of using Sed-Dedup, we decompose the backup time based on the operational steps. In SecDep, the backup time includes key & tag generation time, split time, search time, data encryption time and transmission time. In Sed-Dedup, it consists of key & tag generation time, searching time, encryption time, delta time and transmission time. Here, delta time represents the time consumed to generate the delta file using our delta encoding approach. We evaluate the performance with the average chunk sizes of 2 kB, 4 kB, and 8 kB under the same two scenarios: Two-Versions and Three-Versions. The results are displayed in Figure 5A and Figure 5B.

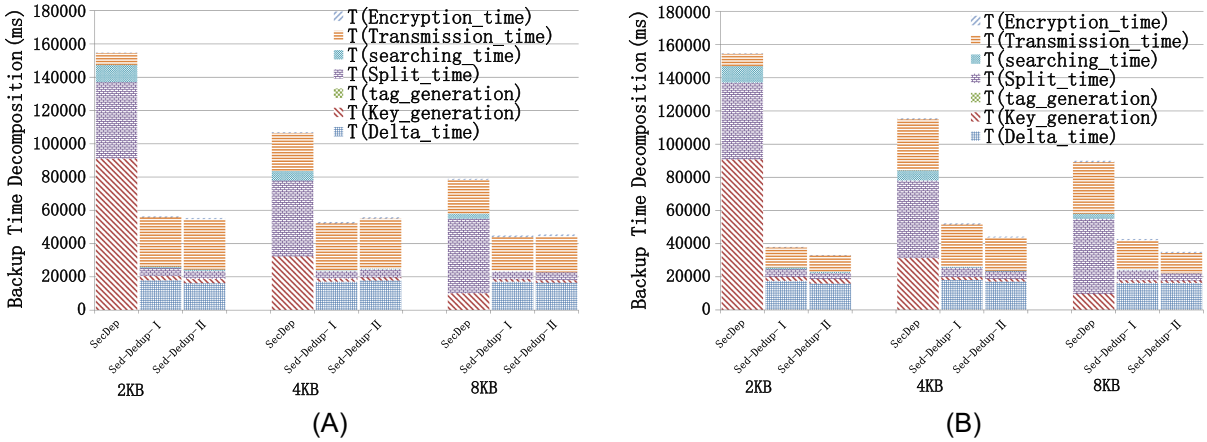

FIGURE 5 Backup time decomposition. A, Backup time decomposition under two-versions scenario; B, Backup time decomposition under three-versions scenario

As we can discover, in both scenarios, Sed-Dedup outperforms SecDep. In Figure 5A, when the average chunk size is 2 kB, the backup time in Sed-Dedup is only 36.48% of that in SecDep. For larger average chunk sizes, SecDep achieves better performance as the accumulative backup time decreases. However, it is still much higher than Sed-Dedup. The most time consuming steps in SecDep are key generation, split and transfer times. Apparently, a large portion of the time is wasted on unnecessarily generated new chunks which contain vastly overlapped data. While in Sed-Dedup, we avoid this problem with the delta files. It greatly reduces the key generation time. Even though it adds the delta time and key generation time, it is still much shorter than the key generation overhead in SecDep. It is because each version in Sed-Dedup needs to be split into chunks and each chunk in Sed-Dedup will generate a chunk-level encryption key while only the delta file's ciphertext recording the difference of consecutive version in Sed-Dedup will be uploaded into the cloud. Moreover, unlike the SecDep, Sed-Dedup will utilize the delta encoding process to record the difference among version files instead of splitting modified version files into chunks.

In Figure 5B, Sed-Dedup-II achieves even more performance improvement under Three-Versions scenario than Sed-Dedup-I. Furthermore, the costs for splitting, transferring and encrypting the data also decrease. In Figure 5B, when chunk size is 4 kB, the key generation cost in Sed-Dedup's two schemes are 5.87% and 5.22% of that in SecDep, respectively. For splitting cost time, Sed-Dedup-I is 11.92% of that in SecDep, while Sed-Dedup-II is 78.99% of that in Sed-Dedup-I. The transmission time in Sed-Dedup-II is 78.61% of that in Sed-Dedup-I. Both schemes in Sed-Dedup are one order less than that of in SecDep for the chunk tag check time.

As we can observe that, the majority time spend in Sed-Dedup is on delta operations. Thus, it is determined by the amount of the modified data volume and does not change too much no matter what chunk size the system uses.

5.4 | File restore time comparison

We also evaluate the file restore time. In Sed-Dedup, two conditions may occur. If F_O is not stored on the local storage, Sed-Dedup has to download it from the SP in addition to the delta files. We consider this as the worst case. If it is already stored locally, Sed-Dedup only retrieve the delta files. We denote it as the general case.

Table 6 and Table 7 show the comparison results. SecDep has slightly better performance than the worst case in Sed-Dedup. This is because Sed-Dedup has to download F_O and the delta files while the only data to be retrieved in SecDep is the modified file. However, Sed-Dedup greatly reduce the restoring time in the general case. Since the user already has the F_O, the system only need to download the delta files. As shown in Table 7, as the version count increases, Sed-Dedup-II consumes more time than Sed-Dedup-I. This is because Sed-Dedup-II needs to download more delta files compared with Sed-Dedup-I as the version count increases. Sed-Dedup-I only need one delta file no matter how many versions are generated. However, both of them outperform SecDep.

5.5 | Key management overhead

Besides the data storage overhead, the key management overhead is not negligible in secure deduplication systems. In this set of experiments, we evaluate the number of keys need to be generated as the number of modified files increases. The result is shown in Figure 6. As we can observe from the results, as the number of modified files increases, the number of keys caused by modified files in SecDep increases from 100 288 for the first new Linux kernel version to 229 022 for the sixth version. For mysql dump files, it increases from 135 285 to 467 014. We can expect that this number increases linearly as more versions are added. While in Sed-Dedup, both two schemes generate less keys. The file-level keys in Sed-Dedup are about the same with SecDep. However, the number of chunk level keys are far less in Sed-Dedup since the introduction of delta files generates much fewer chunks compared with the original file. Apparently, Sed-Dedup can greatly reduce the key generation and management overhead.

TABLE 6 A single file restore operation comparison under two-versions scenario

File Size	32 kB	128 kB	512 kB	2 M	8 M	32 M	128 M
SecDep (ms)	1624	1920	3201	7270	21 797	74 782	311 689
Sed-Dedup-I General Case (ms)	55	52	158	574	1956	4910	25 589
Sed-Dedup-I Worst Case (ms)	1677	2066	3470	8194	24 570	81 195	348 956
Sed-Dedup-II General Case (ms)	51	59	148	599	1903	4992	25 763
Sed-Dedup-II Worst Case (ms)	1675	1979	3349	7869	23 700	79 774	337 452

TABLE 7 A single file restore operation comparison under three-versions scenario

File Size	32 kB	128 kB	512 kB	2 M	8 M	32 M	128 M
SecDep (ms)	1490	1999	3248	7754	22 644	76 457	335 687
Sed-Dedup-I General Case (ms)	53	146	269	924	2773	6413	37 267
Sed-Dedup-I Worst Case (ms)	1543	2145	3517	8621	25 411	82 753	364 523
Sed-Dedup-II General Case (ms)	70	160	328	1126	3209	7687	56 284
Sed-Dedup-II Worst Case (ms)	1560	2159	3576	8823	25 847	84 027	383 540

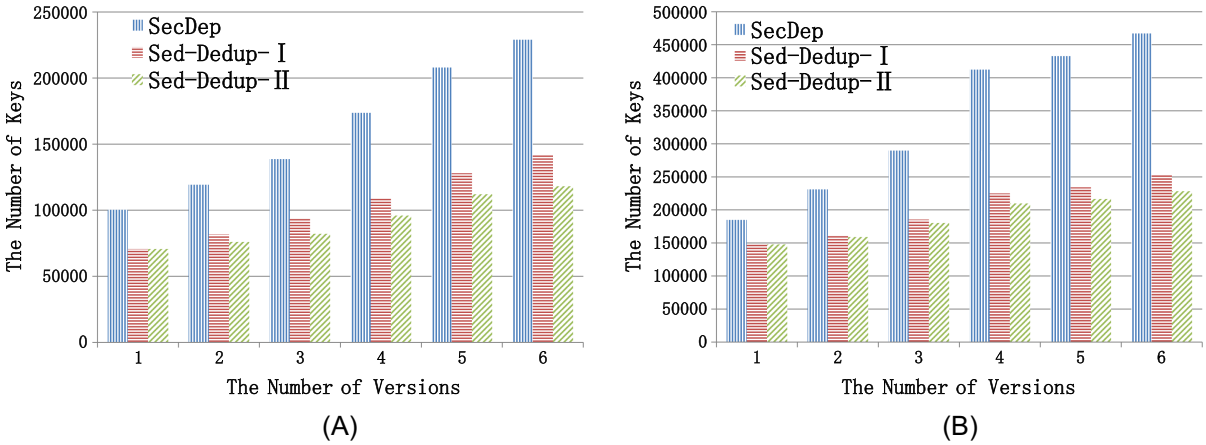

FIGURE 6 Number of encryption keys comparison. A, Backup multiple Linux kernel versions; B, Backup multiple Mysql dump files

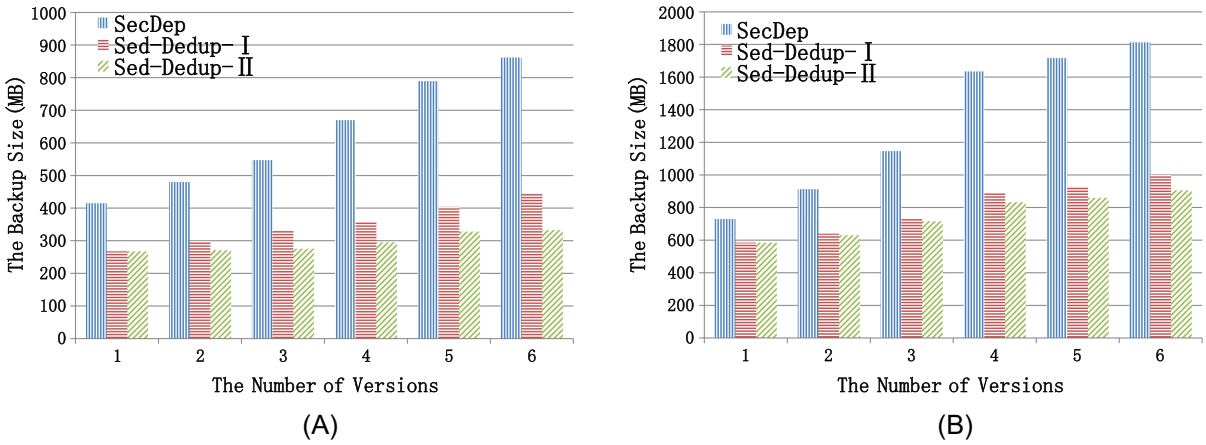

FIGURE 7 Storage space consumption comparison. A, Backup multiple Linux kernel versions; B, Backup multiple Mysql dump files

5.6 | Real world dataset evaluation

5.6.1 | Storage space consumption

In this experiment, we choose the second and third datasets for evaluation. We select 7 Linux kernel versions, the lowest numbered version is chosen as an original file set. For mysql dump files, we also choose the lowest version as the original file and others as modified versions. Here, we assume an original version is already stored on the SP, and case 1 represents that we backup one more version, case 2 means we backup two more versions, and so on.

Figure 7 shows the total backup storage consumption comparison results. Sed-Dedup and SecDep have tremendous performance difference. The backup size due to the modified files in Sed-Dedup increases much slower than SecDep. When there is one more kernel version stored, the backup size in Sed-Dedup-I is increased by 40 MB while it is increased by 188 MB in SecDep. The difference becomes wider as more kernel versions need to be stored. With 6 modified versions, the increased backup size is 218.43 MB in Sed-Dedup-I while it is 635.47 MB in SecDep. Clearly, this problem is caused by the inefficient chunking algorithm used in SecDep. For mysql dump files, the backup size in Sed-Dedup-I increases 49.44 MB, while it is increased by 193.63 MB in SecDep. As the version count increase to 6, the added backup size is 467.18 MB in Sed-Dedup-I while it is 1277.42 MB in SecDep. Using Sed-Dedup-II can achieve even better performance. The final backup size in Sed-Dedup-II is 112.01 MB smaller than Sed-Dedup-I for Linux kernel datasets. For mysql dump file datasets, the number is 97.23 MB.

5.6.2 | Storage time comparison

Figure 8 shows the backup time comparison results. Similar to the storage consumption results, Sed-Dedup outperforms SecDep in all scenarios. Furthermore, Sed-Dedup-II is much faster compared with Sed-Dedup-I. Based on the above results, we can conclude that Sed-Dedup achieves better performance than SecDep for the real world datasets. There are two factors for this improvement. First, Sed-Dedup only needs to keep the modified portions in delta files, it contains no or few duplications comparing with the previous version. Consequently, it can not only avoid the large number of chunking generation overhead, but also reduce the transferring time. While in SecDep, this is not the case. Second, SecDep takes extra time for new encryption key generation, while in Sed-Dedup-I, the number of new encryption keys is much lower. Thus, we can greatly decrease this overhead. Sed-Dedup-II is even faster because it slows down the growth of keys even further.

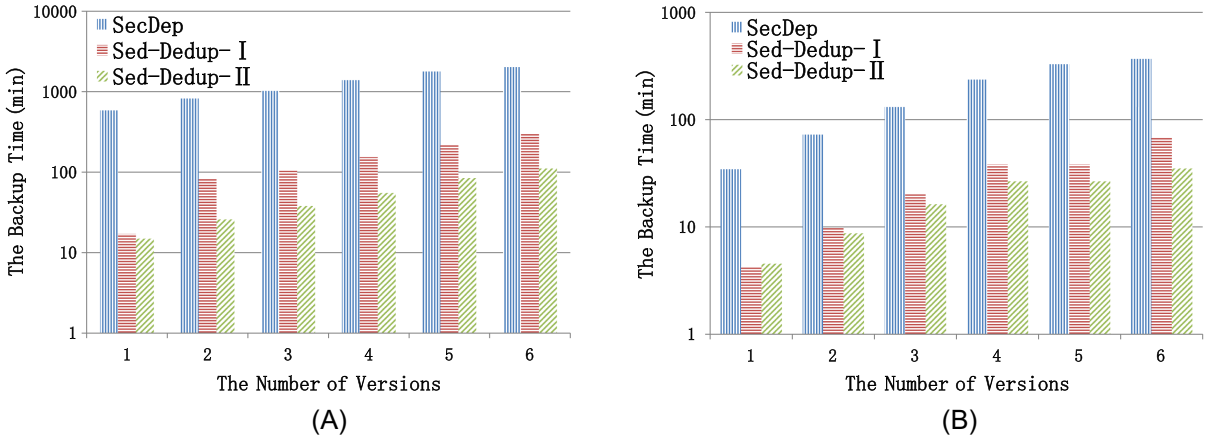

FIGURE 8 Storage time consumption comparison. A, Backup multiple Linux kernel versions; B, Backup multiple Mysql dump files

6 | SECURITY DISCUSSION

In this section, we analyze the data and key security property in Sed-Dedup. We assume the underlying encryption and hash algorithms such as CE and SSSS are secure.

In reality, an adversary may try to compromise the SP or collude with users. When it happens, the chunks on the SP are accessible to the adversary. Thus, it can conduct the brute-force attacks to extract plaintext. Assume the adversary knows the encryption algorithm, once it attains the knowledge that some particular chunks might from a specific plaintext set S, it can encrypt each chunk in S to get the ciphertext and compare with the ciphertext obtained from the SP. If there is a match, the adversary knows the plaintext of the corresponding chunk. It can continue further attacks based on partial results. When the adversary colludes with some legitimate users, it may perform the duplicate-faking attacks on the data by uploading its own chunks which have correct tags but wrong contents.[5]

Sed-Dedup can avoid the above problems and ensures data confidentiality. When a user backups new files, each chunk is encrypted with the corresponding chunk-level key before uploading to the SP. Since each chunk-level key is generated by the content of chunks, breaking the confidentiality of the user's data after delta encoding is extremely difficult. Sed-Dedup can resist duplicate-faking attacks by comparing chunk tags since each chunk also has a tag, a hash value of the ciphertext of chunks. When a user backups a modified file, the brute-force attacks and duplicate-faking attacks can also be prevented. Instead of uploading the ciphertext of the modified file, Sed-Dedup uploads the chunks of delta files onto the SP. All of the chunk-level keys are encrypted by the corresponding file-level key. Therefore, it is difficult to break the confidentiality of user's data, even though an adversary compromises with the SP. For the duplicate-faking attacks, each chunk-level key is encrypted by the file-level key. Each delta file also has a modified file tag, generated by the ciphertext of the modified file. Therefore, the user can resist the duplicate-faking attacks by comparing hash value with its tag. Furthermore, the adversary also need to restore the plaintext of the original file when they want to get the plaintext of the modified file, it makes the attack even harder.

For key security, all the three types of keys including chunk-level keys, file-level keys and shared-level keys need to be considered. All the chunk-level keys are encrypted by the corresponding file-level keys before stored on the SP and all the file-level keys are divided into multiple share keys distributed on DKS. The security guarantees in SecDep also holds in Sed-Dedup.

7 | RELATED WORK

Data deduplication is a special compression technique to reduce the storage space consumption. Thus, it has been widely used in archival and backup systems to improve space utilization efficiency.[8] Venti[37] is a network storage system which applies the unique hash value as the block identifier for deduplication to prevent accidental or malicious destruction of data. However, it is not suitable for large scale data applications. Hong et al[38] designed a duplication elimination algorithm. It was running as a background process in a SAN file system. By integrating content hashing, copy-on-write, and lazy update technique, it can improve system performance and save storage occupation. MAD2[39] proposed a scalable high throughput deduplication mechanism for network backup services by eliminate duplicate data at the file-level and the chunk-level. To achieve high performance, MAD2 focused on the hash bucket matrix organization and used Bloom Filter Array as a quickly indicator to find a possible duplicate. The above systems determine data duplications based on the locality check. However, they were not working well if not sufficient locality can be discovered.

Other systems use different approaches. Extreme Binning[40] presented a scalable deduplication technique. It can discover data duplication based on similarity instead of locality. PeerDedupe[41] proposed a novel scheme called PeerDedupe to conduct an in-depth and quantitative investigation on the peer-assisted deduplication. It is designed for mainstream server-side deduplication. In addition, many other deduplication

system architecture have been developed such as Opendedup,[42] and lessfs,[43] etc. However, None of them considered the security problem during deduplication processs.

Secure data deduplication preserves the confidentiality of outsourced data by adding encryption functionality. Most secure deduplication systems adopt Convergent Encryption (CE)[44] to enforce the data security while making cross-user data duplication checks feasible. It is a deterministic encryption mechanism.[5] utilizes the message-locked encryption to protect data confidentiality by transforming the predictable messages into unpredictable messages. It proposes system architecture to use third party key servers, and generates file tags for data duplication checks. Jia et al[45] address security concerns in cross-user deduplication of encrypted files in the cloud storage. They enhance and generalize the CE method on their own efficient hash function with large output sizes.

However, CE method results in key management problems since each file or chunk has a separate encryption key. To solve this problem, Clouddedup[46] backups the convergent keys on external key servers and develops a secure and efficient storage service. DupLESS[17] uses a key server via RSA-ORPF protocol to manage the keys. However, directly storing the keys on the key server suffers from the simple point of failure problem. Therefore, CEKM[47] proposes a method to break each key with Shamir secret Sharing scheme (SSSS) to generate multiple share keys and distributes them on separate servers to solve the problem. However, in CEKM, the management of the large number of chunk-level keys greatly increases the management overhead.

Liu et al[19] propose a new secure deduplication scheme which supports client-side encryption without any additional independent servers. It is based on password authenticated key exchange mechanism. All the users have a common session key if they share a short secret "password". However, the performance of this scheme decreases sharply when it is applied for block-level deduplication. Tang et al[20] introduce a novel ciphertext deduplication technique. It applies the deduplication methods based on a classical CP-ABE scheme.[48] However, the author did not disclose the specifical deduplication ratio performance comparison with other secure deduplication systems. SecDep[18] designs a secure deduplication system showing that storing the chunk-level keys on the SP can reduce the overhead. To ensure the security, the chunk-level keys are encrypted by the corresponding file-level keys. SecDep creates share keys for each file-level key with SSSS method and distribute them on key servers. Thus, it greatly reduce the number of keys need to be managed on the key servers.

8 | CONCLUSIONS AND FUTURE WORK

Secure deduplication systems become increasingly popular as they can ensure the security and privacy of outsourced data. However, the state-of-the-art secure deduplication mechanisms cannot deal with frequent file modification problems efficiently and result in unnecessary operational overheads. In this paper, we present Sed-Dedup, an efficient secure data deduplication system to address this issue. In Sed-Dedup, a novel delta encoding protocol is introduced which only records the difference between two versions in a delta file. With this approach, Sed-Dedup avoids generating largely overlapped chunks. Thus, Sed-Dedup achieves higher deduplication ratio, smaller cloud storage consumption, and lower network traffic. Furthermore, Sed-Dedup can greatly decrease the key management overhead. To evaluate its performance, we compare Sed-Dedup with SecDep using synthetic and real world datasets. The experimental results clearly show that Sed-Dedup can achieve superior performance.

In our current experiments, we choose Linux kernel source code archives and mysql dump files as the real world backup system datasets to evaluate the performance. It might not be the best option. In the future, we plan to collect real world datasets as described in the work of Zhou et al[18] and use them for evaluation. We also plan to contact cloud service providers for real backup system datasets. In addition, we plan to further investigate the advantages and disadvantages of delta encoding techniques, and design more efficient algorithms to keep track the difference between multiple version files.

ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China under grants 2016YFB0800402 and 2016QY01W0202; National Natural Science Foundation of China under grants U1836204, 61572221, 61433006, U1401258, and 61502185; Major Projects of the National Social Science Foundation under grant 16ZDA092; and Guangxi High Level Innovation Team in Higher Education Institutions—Innovation Team of ASEAN Digital Cloud Big Data Security and Mining Technology.

ORCID

Wenlong Tian https://orcid.org/0000-0003-3177-9099

REFERENCES

1. Google Drive. A file storage and synchronization service developed by Google. https://www.google.com/drive
2. Microsoft OneDrive. A file hosting service and synchronization service operated by Microsoft. https://onedrive.live.com/
3. Baidu Cloud. A cloud storage service provided by Baidu. https://yun.baidu.com/

4. Meyer DT, Bolosky WJ. A study of practical deduplication. Paper presented at: Usenix Conference on File and Storage Technologies; 2012; San Jose, CA.
5. Wallace G, Douglis F, Qian H, et al. Characteristics of backup workloads in production systems. Paper presented at: FAST'12:4–4 USENIX Association; 2012; Berkeley, CA.
6. Dropbox. A personal cloud storage service for file sharing and collaboration. http://www.dropbox.com
7. Dell EMC. An online backup service for both Windows and macOS users provided by Dell EMC. http://www.mozy.com
8. João P, Orlando PJ. A survey and classification of storage deduplication systems. *ACM Comput Surv*. 2014;47(1):1-30.
9. Ng WK, Wen Y, Zhu H. Private data deduplication protocols in cloud storage. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing; 2012; New York, NY.
10. Bugiel S, Nürnberger S, Sadeghi A-R, Schneider T. Twin clouds: secure cloud computing with low latency. In: *Communications and Multimedia Security: 12th IFIP TC 6/TC 11 International Conference, CMS 2011, Ghent, Belgium, October 19-21, 2011. Proceedings*. Berlin, Germany: Springer; 2011:32-44. *Lecture Notes in Computer Science*.
11. Howard R. Data encryption standard. *Comput Secur*. 1997;6(3):195-196.
12. Miller FP, Vandome AF, McBrewster J. *Advanced Encryption Standard*. Orlando, FL: Alpha Press; 2009.
13. Li J, Chen X, Li M, Li J, Lee PPC, Lou We. Secure deduplication with efficient and reliable convergent key management. *IEEE Trans Parallel Distrib Syst*. 2014;25(6):1615-1625.
14. Bellare M, Keelveedhi S, Ristenpart T. Message-locked encryption and secure deduplication. Paper presented at: Annual International Conference on the Theory and Applications of Cryptographic Techniques; 2013; Athens, Greece.
15. Eastlake D 3rd, Jones P. US secure hash algorithm 1 (SHA1). RFC 3174. 2001.
16. Halevi S, Harnik D, Pinkas B, Shulman-Peleg A. Proofs of ownership in remote storage systems. In: Proceedings of the 18th ACM Conference on Computer and Communications Security; 2011; Chicago, IL.
17. Keelveedhi S, Bellare M, Ristenpart T. DupLESS: Server-aided encryption for deduplicated storage. Paper presented at: 22nd USENIX Security Symposium; 2013; Washington, DC.
18. Zhou Y, Feng D, Xia W, et al. SecDep: A user-aware efficient fine-grained secure deduplication scheme with multi-level key management. Paper presented at: 2015 31st Symposium on Mass Storage Systems and Technologies (MSST); 2015; Santa Clara, CA.
19. Liu J, Asokan N, Pinkas B. Secure deduplication of encrypted data without additional independent servers. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security; 2015; Denver, CO.
20. Tang H, Cui Y, Guan C, Wu J, Weng J, Ren K. Enabling ciphertext deduplication for secure cloud storage and access control. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security; 2016; Xi'an, China.
21. Meister D, Brinkmann A. Multi-level comparison of data deduplication in a backup scenario. In: Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference; 2009; Haifa, Israel.
22. GIT Version Control. https://lab.github.com/docs
23. Linux Kernel Archives. https://kernel.org/
24. Hunt JJ, Vo K-P, Tichy WF. Delta algorithms: an empirical analysis. *ACM Trans Softw Eng Methodol*. 1998;7(2):192-214.
25. Kulkarni P, Douglis F, LaVoie JD, Tracey JM. Redundancy elimination within large collections of files. Paper presented at: USENIX Annual Technical Conference; 2004; Boston, MA.
26. Shilane P, Wallace G, Huang M, Hsu W. Delta compressed and deduplicated storage using stream-informed locality. Paper presented at: USENIX Workshop on Hot Topics in Storage and File Systems; 2012; Boston, MA.
27. Rabin M. *Fingerprinting by Random Polynomials*. Technical Report. Cambridge, MA: Center of Research in Computer Technology, Harvard University; 1981.
28. Kave E, Khuern TH. *A Framework for Analyzing and Improving Content-Based Chunking Algorithms*. Technical Report. Palo Alto, CA: Hewlett-Packard Labs; 2005.
29. Kruus E, Ungureanu C, Dubnicki C. Bimodal content defined chunking for backup streams. Paper presented at: 8th USENIX Conference on File and Storage Technologies; 2010; San Jose, CA.
30. Zhu B, Li K, Patterson H. Avoiding the disk bottleneck in the data domain deduplication file system. Paper presented at: 6th USENIX Conference on File and Storage Technologies; 2008; San Jose, CA.
31. Xia W, Jiang H, Feng D, Hua Y. SiLo: A similarity-locality based near-exact deduplication scheme with low RAM overhead and high throughput. Paper presented at: 2011 USENIX Annual Technical Conference; 2011; Portland, OR.
32. Lu G, Nam YJ, Du DHC. BloomStore: Bloom-filter based memory-efficient key-value store for indexing of data deduplication on flash. Paper presented at: IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST); 2012; San Diego, CA.
33. Datar M, Immorlica N, Indyk P, Mirrokni VS. Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the Twentieth Annual Symposium on Computational Geometry; 2004; Brooklyn, NY.
34. Chuanshuai Y, Zhang C, Mao Y, Li F. Leap-based content defined chunking—theory and implementation. Paper presented at: 2015 31st Symposium on Mass Storage Systems and Technologies (MSST); 2015; Santa Clara, CA.
35. Zhang Y, Jiang H, Feng D, et al. AE: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication. Paper presented at: 2015 IEEE Conference on Computer Communications (INFOCOM); 2015; Hong Kong.
36. Muthitacharoen A, Chen B, Maziéres D. A low-bandwidth network file system. *ACM SIGOPS Oper Syst Rev*. 2001;35(5):174-187.
37. Quinlan S, Dorward S. Awarded best paper! - Venti: A new approach to archival data storage. In: Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST'02); 2002; Berkeley, CA.
38. Hong B, Plantenberg D, Long DDE, Sivan-Zimet M. Duplicate data elimination in a SAN file system. Paper presented at: Twenty-first IEEE Conference on Mass Storage Systems and Technologies; 2004; College Park, MA.

39. Wei J, Jiang H, Zhou K, Feng D. MAD2: A scalable high-throughput exact deduplication approach for network backup services. Paper presented at: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST); 2010; Incline Village, NV.
40. Bhagwat D, Eshghi K, Long DDE, Lillibridge M. Extreme Binning: Scalable, parallel deduplication for chunk-based file backup. Paper presented at: 2009 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems; 2009; London, UK.
41. Xing Y., Li Z., Dai Y. PeerDedupe: Insights into the peer-assisted sampling deduplication. Paper presented at: 2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P); 2010; Delft, Netherlands.
42. Global inline deduplication for block storage and files. http://www.opendedup.org
43. Lessfs. Lessfs–open source data de-duplication. https://github.com/crass/lessfs
44. Douceur JR, Adya A, Bolosky WJ, Simon D, Theimer M. Reclaiming space from duplicate files in a serverless distributed file system. In: Proceedings 22nd International Conference on Distributed Computing Systems; 2002; Vienna, Austria.
45. Jia X, Chang E-C, Zhou J. Weak leakage-resilient client-side deduplication of encrypted data in cloud storage. In: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security; 2013; Hangzhou, China.
46. Puzio P, Molva R, Onen M, Loureiro S. ClouDedup: Secure deduplication with encrypted data for cloud storage. Paper presented at: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom2013); 2014; Bristol, UK.
47. Agrawal NO, Kulkarni SS. Secure deduplication and data security with efficient and reliable CEKM. *Int J Appl Innov Eng Manag*. 2014;3:335-340.
48. Waters B. Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization. *Lect Notes Comput Sci*. 2011;2008:321-334.

How to cite this article: Tian W, Li R, Xu C-Z, Xu Z. Sed-Dedup: An efficient secure deduplication system with data modifications. *Concurrency Computat Pract Exper*. 2019;e5350. https://doi.org/10.1002/cpe.5350