SecDep: A User-Aware Efficient Fine-Grained Secure Deduplication Scheme with Multi-Level Key Management

Yukun Zhou, Dan Feng*, Wen Xia, Min Fu, Fangting Huang, Yucheng Zhang, Chunguang Li
Wuhan National Laboratory for Optoelectronics
School of Computer, Huazhong University of Science and Technology, Wuhan, China
*Corresponding author: dfeng@hust.edu.cn

***Abstract*—Nowadays, many customers and enterprises backup their data to cloud storage that performs deduplication to save storage space and network bandwidth. Hence, how to perform secure deduplication becomes a critical challenge for cloud storage. According to our analysis, the state-of-the-art secure deduplication methods are not suitable for cross-user fine-grained data deduplication. They either suffer brute-force attacks that can recover files falling into a known set, or incur large computation (time) overheads. Moreover, existing approaches of convergent key management incur large space overheads because of the huge number of chunks shared among users.**

Our observation that cross-user redundant data are mainly from the duplicate files, motivates us to propose an efficient secure deduplication scheme SecDep. SecDep employs User-Aware Convergent Encryption (UACE) and Multi-Level Key management (MLK) approaches. (1) UACE combines cross-user file-level and inside-user chunk-level deduplication, and exploits different secure policies among and inside users to minimize the computation overheads. Specifically, both of file-level and chunk-level deduplication use variants of Convergent Encryption (CE) to resist brute-force attacks. The major difference is that the file-level CE keys are generated by using a server-aided method to ensure security of cross-user deduplication, while the chunk-level keys are generated by using a user-aided method with lower computation overheads. (2) To reduce key space overheads, MLK uses file-level key to encrypt chunk-level keys so that the key space will not increase with the number of sharing users. Furthermore, MLK splits the file-level keys into share-level keys and distributes them to multiple key servers to ensure security and reliability of file-level keys.

Our security analysis demonstrates that SecDep ensures data confidentiality and key security. Our experiment results based on several large real-world datasets show that SecDep is more time-efficient and key-space-efficient than the state-of-the-art secure deduplication approaches.

I. Introduction

With the rapid development of cloud computing, increasing number of users and enterprises would like to backup their data to cloud storage. IDC predicts that digital data will exceed 44ZB in 2020 [1]. Moreover, some recent publications also show that there is a large amount of duplicate data among users in storage systems, especially for backup systems [2, 3]. Data deduplication is a technique to eliminate duplicate copies of data [4], which has been gaining increasing popularity in cloud storage. For example, Dropbox [5], SpiderOak [6], and Mozy [7], have adopted data deduplication to save storage space and network bandwidth. Generally speaking, deduplication eliminates redundant data by keeping only one physical copy that could be referenced by other duplicate data (copies). Deduplication can be implemented at different granularities: a file (i.e., file-level deduplication), or a fine-grained data chunk (i.e., chunk-level deduplication) [8]. The latter has been widely used because it renders the system more flexible and efficient.

To protect data confidentiality, users usually encrypt data to make the data scrambled with their own keys, which makes deduplication impossible. Specifically, the same data encrypted by different users' keys will result in different ciphertexts such that duplicates will not be found. However, sharing keys among users can result in data leakage because of user compromise [9]. Therefore, how to ensure security of users' data is the main problem facing cross-user fine-grained deduplication-based cloud storage systems. Most of existing secure deduplication solutions use a deterministic encryption method called Convergent Encryption (CE) [10]. CE uses a hash of the data as a key to encrypt the data. Hence, CE will encrypt the identical data into the same ciphertext, which enables deduplication on the ciphertext.

Convergent Encryption nevertheless brings new challenges. **First**, it either suffers brute-force attacks [9] or incurs large computation overheads. Specifically, CE suffers brute-force attacks because of deterministic and keyless issues [9]. If the adversary knows the target ciphertext C of the target data D is in a specific or known set $S = \{D_1, ..., D_n\}$ of size n, the adversary can recover the data D from the set S by off-line encryption. For each i = 1, ..., n, the adversary just simply encrypts D_i via CE to get the ciphertext denoted C_i and returns the D_i such that $C = C_i$, which actually breaks the ciphertext C. To solve this problem, Bellare et al. propose DupLESS [9], which encrypts data by a message-locked key obtained from a key-server via RSA-OPRF protocol [11] to resist brute-force attacks. However, DupLESS is inefficient and incurs large computation overheads for chunk-level deduplication. Specifically, each chunk performs the time-consuming RSA-OPRF protocol [9, 11] to generate message-locked key. The key generation for chunk-level deduplication will incur significant computation overheads. The key generation time will increase with the enormous number of chunks.

Second, existing key management approaches of CE also have several limitations, including large key space overheads and single-point-of-failure [12]. Generally, users encrypt their data by the convergent keys and protect convergent keys with

978-1-4673-7619-8/15/$31.00

users' master key [9, 13]. Thus, the number of convergent keys increases linearly with the number of unique data and number of sharing users [12]. Master key suffers from a single point of failure risk [12]. To ensure chunk keys' security and reliability, Dekey [12] splits chunk keys into key shares via Ramp Secret Sharing Scheme (RSSS) [14, 15] and distributes key shares to multiple servers. As a result, the number of convergent keys explodes with the number of key shares.

To overcome the aforementioned challenges, we develop a secure and efficient deduplication scheme called SecDep. The main idea behind SecDep is to exploit redundant data distribution among and inside users, and use variants of CE to make a trade-off between data security and duplicate detection performance. Our key observation is that cross-user redundant data are mainly from the duplicate files, which motivates us to propose User-Aware Convergent Encryption (UACE) and Multi-Level Key management (MLK) approaches. Specifically, **(1)** for an inputed file, UACE performs cross-user deduplication at file-level and file-level keys are generated by server-aided HCE [16–18]. If it is not a duplicate file, UACE performs inside-user chunk-level deduplication with the user-aided CE. The secret information provided by the user is used for generating a more secure chunk-level CE key. **(2)** MLK encrypts chunk-level keys with the corresponding file-level keys. MLK splits file-level keys into share-level keys via Shamir Secret Sharing Scheme (SSSS) [19] and sends them to the Distributed Key Servers.

Security analysis demonstrates that SecDep could resist attacks from both external and internal adversaries. As a result, SecDep significantly reduces both of time and key space overheads compared with the state-of-the-art schemes, while ensuring a comparable high-level data security.

This paper makes the following contributions.

- SecDep proposes a User-Aware Convergent Encryption (UACE) approach to resist brute-force attacks and reduce time overheads. UACE ensures both data security and deduplication efficiency.
- SecDep proposes a Multi-Level Key management (MLK) approach to ensure key security and reduce key space overheads. MLK also provides the confidentiality and reliability of file-level keys by splitting file-level keys into share-level keys and distributing them to multiple key servers.
- Our security discussion demonstrates that SecDep is secure under the proposed threat model. SecDep achieves security goals and ensures the security of both users' data and keys by using our UACE and MLK approaches.
- We implement a prototype of SecDep via UACE and MLK approaches. Experimental results based on four real-world datasets suggest that SecDep reduces 52-92% of time overheads at the expense of only losing 2.8-7.35% of dedup factor compared with the sate-of-the-art DupLESS-chunk approach. SecDep reduces 34.8-96.6% of key space overheads compared with the state-of-the-art Dekey and Master Key approaches.

The rest of paper is organized as follows. Section II presents the background and motivation of this paper. Section III describes the threat model and security goals. Section IV presents the design and workflow of SecDep. Section V describes the implementation details. In Section VI, we discuss the security of SecDep. Section VII describes our experimental evaluation of SecDep and the performance comparisons among the state-of-the-art approaches. Finally, we draw conclusion in Section IX.

II. Background & Motivation

In this section, we will describe the cryptographic primitive of Convergent Encryption, which is the most widely used encryption scheme for data deduplication. In addition, we will analyze and make comparison of existing secure deduplication methods and key management approaches. Finally, we will discuss several key observations that motivate our work.

A. Convergent Encryption for Data Deduplication

For data deduplication based storage systems, the same data encrypted with different users' keys will generate different ciphertexts, which makes deduplication impossible for eliminating cross-user redundant data. To solve this problem, previous work formalizes the primitive of Convergent Encryption [10] or Message-Locked Encryption [18]. Convergent Encryption is presented as five-tuple polynomial-time algorithms listed below. A user generates a convergent key via Genkey from the original data copies and encrypts them with the key by Encry. The tag will be generated via GenTag and can be used for duplicate checking. If two data have the same tag, they are considered as identical. Tag is also referred to as "fingerprint" in other papers [8]. We give some definitions of the five tuples as follows [10, 18].

- Setup(1^λ) $\longrightarrow P$. P is the public parameter that is generated by using a parameter generation algorithm [9] with the security parameter 1^λ. For all $\lambda \in \mathbb{N}$,P$\in [Setup(1^\lambda)]$.
- GenKey(P, M) $\longrightarrow K$. K is a message-derived key that is generated by a key generation algorithm. And the public parameter P and a message (i.e., data) M are inputs. K is also called convergent key.
- Encry(P, K, M) $\longrightarrow C$. Encry is a deterministic and symmetric encryption algorithm. C is the ciphertext of message M returned by Encry. And the public parameter P, message-derived key K, and message M are inputs.
- GenTag(P, C) $\longrightarrow T$. GenTag is the tag generation algorithm. It uses public parameter P and ciphertext C as inputs and returns data tag T for duplicate checking.
- Decry(P, K, C) $\longrightarrow M$. Decry is also a deterministic and symmetric decryption algorithm that takes public parameter P, convergent key K, and data ciphertext C ($C \in \{0,1\}^*$) as inputs. Finally, the returned M is the decrypted data.

B. State of The Art on Secure Deduplication

Focusing on data security and privacy, existing secure deduplication methods suffer mainly two problems: data confiden-

TABLE I
STATE OF THE ART ON SECURE DEDUPLICATION. 'F' AND 'B' REPRESENT FILE-LEVEL AND CHUNK-LEVEL DEDUPLICATION RESPECTIVELY.

Security Goals	Approaches	Granularity	Representative Work	Limitations
Data Confidentiality	CE	F & B	Farsite [10], Bellare et al. [18]	Brute-force atttacks
	HCE	F & B	Pastiche [16], Storer et al. [20], MLE [18]	Brute-force attacks, duplicate-faking attacks
	DupLESS	F	DupLESS [9]	Large computation overheads (chunk-level)
Key Security	Single Key Server	F & B	Storer et al. [20], ClouDedup [13]	Single point of failure
	Master Key	F & B	Pastiche [16], Dekey [12], DupLESS [9]	Single point of failure, key space overheads
	Secret Splitting	B	Dekey [12]	Large key space overheads

tiality and key security. State-of-the-art secure deduplication methods and key management approaches are summarized in table I. We will analyze these approaches and their limitations in detail as below.

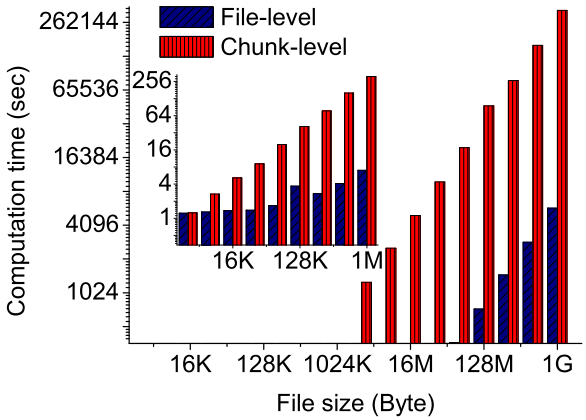


Fig. 1. The computation time of RSA-OPRF protocol on file-level and chunk-level deduplication. Note that the later uses the average chunk size of 4KB.

To ensure data confidentiality, existing secure deduplication methods mainly include: Convergent Encryption (CE) [10, 18], Hash Convergent Encryption (HCE) [16, 18], and DupLESS [9]. First, Convergent Encryption, where data is encrypted with it's hash value and tag is generated from ciphertext of data, has been adopted by Farsite [10]. However, Convergent Encryption is vulnerable to brute-force attacks because it is deterministic and keyless [9]. If the adversary knows the target data is in a specific or known set, he could encrypt data to generate ciphertext and compare with target ciphertext to recover the original data. Second, Hash Convergent Encryption (HCE), where data is encrypted with it's hash value and the tag is generated by hashing encryption key, is proposed by Pastiche [16]. HCE offers better performance than CE, because HCE just simply hashes the encryption key while CE needs to first encrypt the entire file and then hash the ciphertext. Nevertheless, HCE suffers brute-force attacks and duplicate-faking attacks [18], where a legitimate message is undetectable replaced by a fake one. Third, Bellare et al. [9] propose DupLESS which encrypts data with message-based keys obtained from a key-server via an oblivious PRF protocol (i.e., RSA-OPRF) [11]. DupLESS incurs large computation time overheads on fine-grained deduplication because oblivious PRF protocol is time-consuming.

In Figure 1, we test and compare the time consumptions of RSA-OPRF protocol for file-level and chunk-level deduplication. The results demonstrate that DupLESS is inefficient and not suitable for fine-grained deduplication. This is because the modular exponentiation in RSA-OPRF protocol is time-consuming, which incurs large computation overheads for chunk-level deduplication due to a large number of chunks. The time overhead of chunk-level deduplication will increase with the number of chunks compared with file-level deduplication.

To ensure key security, there are three approaches for fine-grained convergent key management, Single Key Server, Master Key, and Secret Splitting. **(1)** Single Key Server, which stores all keys on a key server, has adopted by Storer et al. [20], ClouDedup [13], etc. However, the key server suffers from a single point of failure risk. If the key server is compromised, all users' keys can be stolen by adversaries. **(2)** Master Key is an approach that encrypts chunk-level keys with users' own master key and stores the master keys securely. Nevertheless, it is not suitable for fine-grained deduplication because the number of convergent key will increase with the number of users and unique chunks [12]. In addition, Master key suffers from a single point of failure risk [12]. **(3)** Dekey [12] divides each chunk-level key into key shares and distributes pieces to different key servers. However, Dekey's goal is to ensure the reliability of convergent keys, but not the key space efficiency. And hence, Dekey still suffers enormous key space overheads.

C. Observation & Motivation

As shown in Table I, existing encryption solutions for fine-grained deduplication either suffer brute-force attacks, or incur large time overheads. State-of-the-art key management approaches have limitations, including large key space overheads and single-point-of-failure. In order to solve the above challenges, it is necessary to propose a new secure deduplication scheme with an efficient key management approach to balance deduplication security and efficiency. In table II, we study the data redundancy distribution on the four large real-world datasets (whose workload characteristics are detailed in Table VI in Section VI). "Total" represents the duplicates eliminated by global (i.e., cross- & inside-user) chunk-level deduplication. "Cross-user file-level" represents the duplicates eliminated by cross-user file-level deduplication. "Inside-user chunk-level" represents the duplicates eliminated by the inside-user (chunk-level) after (cross-user) file-level deduplication. According to the results shown in Table II, we obtain two key observations.

TABLE II
THE DATA REDUNDANCY DISTRIBUTION ON FOUR LARGE REAL-WORLD DATASETS.

Datasets	Total (global)	Cross-user dup files	Inside-user dup chunks	Cross-user dup chunks
One-set (GB)	202.1 (100%)	104.28 (51.6%)	78.62 (38.9%)	19.2 (9.5%)
Inc-set (GB)	141.3 (100%)	108.8 (77%)	27.13 (19.2%)	5.37 (3.8%)
Full-set (GB)	2457.6 (100%)	2393.7 (97.4%)	60 (2.4%)	1.95 (0.02%)
FSLhomes (GB)	14463.3 (100%)	13764.7 (95.17%)	687 (4.85%)	11.6 (0.08%)

TABLE III
THE COMPARISONS OF DIFFERENT SECURE DEDUPLICATION SCHEMES IN TERMS OF DEDUPLICATION EFFICIENCY AND SECURITY OVERHEADS.

Deduplication scheme	High dedup factor	Low security time overheads
Global dedup at chunk-level	$\surd$	$\times$
Cross-user dedup at file-level	$\times$	$\surd$
Inside-user dedup at chunk-level	$\times$	$\surd$
SecDep	$\surd$	$\surd$

- Cross-user redundant data are mainly from the duplicate files. Previous work SAM [21] and Microsoft's study [2], have similar observations. Meanwhile, there are remaining substantial duplicate chunks inside users. Therefore, the combination of cross-user file-level and inside-user chunk-level deduplication can achieve a comparable performance of redundancy elimination with the global chunk-level deduplication, which can be exploited to implement a more efficient security scheme.
- Cross-user and inside-user deduplication schemes face different security challenges. Specifically, cross-user deduplication generally requires a secure method with high overheads, for example, the DupLESS solution [9]. As shown in Figure 1, chunk-level deduplication using the RSA-OPRF protocol incurs significantly larger time overheads than file-level deduplication. Moreover, inside-user deduplication could employ a more efficient secure method. Motivated by this observation, we employ different secure policies for cross- and inside-user deduplication to balance data security and deduplication performance.

In Table III, global chunk-level deduplication achieves a high dedup factor, but it brings huge security overheads. Cross-user file-level and inside-user chunk-level deduplication provide security with low overheads, but they achieve a low dedup factor. File-level cross-user deduplication approach cannot eliminate all redundant data. Comparing with the global chunk-level deduplication, file-level cross-user deduplication approach misses to detect some duplicate chunks both among and inside users. Both of these schemes alone are not suitable for fine-grained secure deduplication. In order to achieve a high dedup factor and ensure data security, we propose SecDep that combines cross-user file-level and inside-user chunk-level secure deduplication to eliminate more redundant data. SecDep employs server-aided HCE at file-level deduplication and user-aided CE at chunk-level deduplication to ensure data security, while significantly reducing security overheads.

TABLE IV
ACRONYMS USED IN THIS PAPER

Acronym	Description
CE	Convergent Encryption
HCE	Hash Convergent Encryption
SP	Storage Provider
DKS	Distributed Key Servers
$SSSS(w,t)$	Shamir Secret Sharing Scheme with the parameter w and t

III. SYSTEM MODEL & SECURITY GOALS

In this section, we first present the system model of SecDep, a secure and efficient deduplication system with multi-level key management for cloud backups. Then we state the threat model and security goals of SecDep. Some important acronyms used in this paper are listed in Table IV.

A. System Model

As shown in Figure 2, our system SecDep consists of Users, a Storage Provider (SP), and Distributed Key Servers (DKS). When a user wants to access the DKS and the SP, his/her passwords and credentials should be verified at first. Chunks and chunk-level keys are encrypted and stored on the SP. File-level keys are securely divided into share-level keys via Shamir Secret Sharing Scheme (SSSS) [19]. Share-level keys are stored separately on the DKS. Data stored on the DKS and the SP are protected by some access control policies. Specifically, these entities are connected via enterprise network and secure data transmission among them is ensured by the well-known secure communication protocol (such as SSL/TLS) [22].

- **User**. A user is an entity who wants to upload data to (download data from) the Storage Provider (SP). The user applies variants of CE to protecting the data chunks and keys. To resist brute-force attacks and provide data confidentiality, the user accesses the DKS to add secret for generating the random file-level keys, which is detailed in subsection IV-B.
- **Storage Provider (SP)**. The storage providers mainly offer computation and storage services. The SP maintains tag indices for chunk-level and file-level duplicate checking. The SP also stores ciphertexts of chunks and chunk-level keys, which is detailed in subsection IV-C.
- **Distributed Key Servers (DKS)**. The DKS is built on a quorum of key servers via Shamir Secret Sharing Scheme (SSSS) [19] to ensure security of keys. The user splits file keys into w shares via SSSS (w, t) and any t ($\leq w$) of shares can recover file key. Each key server is a stand-alone entity that adds secret for key generations and stores users' key shares, which is detailed in subsection IV-C.

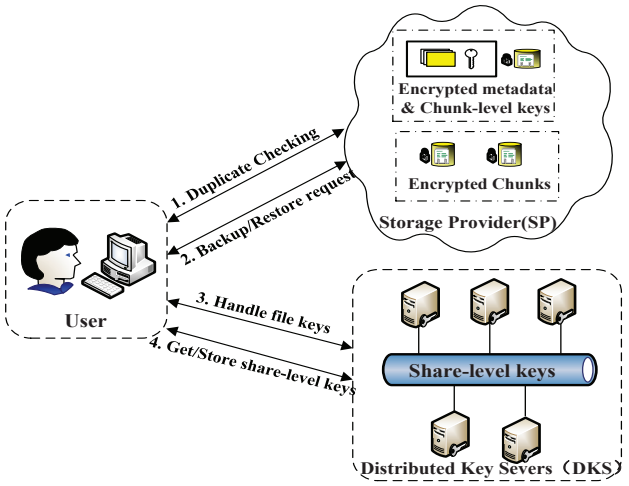


Fig. 2. The system model of SecDep, which contains three entities. Users interact with one or more key servers, and backup/restore data from the SP.

B. Threat Model and Security Goals

Threat Model. The goal of adversaries is to obtain data or keys which don't belong to them. We consider two types of adversaries: internal adversaries and external adversaries. **i)** Internal adversaries try to access the DKS and the SP. **ii)** External adversaries only access the DKS and the SP with communication channels. We assume that both of these adversaries follow the protocol and access control policies. We only focus on internal adversaries, because internal adversaries have much more attack capacities than external adversaries. The adversary can compromise the SP and collude with users. The adversary also tries to steal data from the DKS. The detailed security analysis against adversaries and attacks are discussed in Section VI.

- **Compromising the SP.** We assume that the SP is untrusted and insecure. The adversary tries to compromise the SP. The adversary gets the ciphertext of chunks and chunk-level keys that do not belong to them. For a file belonging to a specific user, the adversary tries to obtain the original file via performing brute-force attacks.
- **Colluding with users.** We assume that the adversary colludes with some users. The adversary seeks to get data or keys owned by other legitimate users. The adversary performs duplicate-faking attacks with these users.
- **Stealing data from the DKS.** We assume that the adversary steals key shares from a predefined number (less than t) of key servers. The adversary tries to recover the file key or obtain some useful data.

Security Goals. We address problems of data confidentiality and key security for cross-user fine-grained deduplication-based cloud backups. Our security goals are listed as below.

- **Data confidentiality**. We need to assure the security of encrypted data copies in the SP. Specifically, the data copies are secure to resist brute-force attacks. We also protect the integrity of chunks that are uploaded to the SP and resist duplicate-faking attacks.

TABLE V
NOTATIONS USED IN THIS PAPER

Notation	Description
(N, e)	Public key of RSA
(N, d)	Private key of RSA
F	File
B	Chunk or block of file
H_F	File hash for key generation
K_F	File-level key
T_F	File tag for duplicate checking
S_F	Share-level keys
K_B	Chunk-level or block-level key
T_B	Chunk or block tag
IsDup	Results of file-level duplicate checking
$\psi[i]$	Results of chunk-level duplicate checking
C_{data}	Ciphertext of chunk
C_{key}	Ciphertext of chunk-level key
Hash	Hash function
HMAC	Hash function with a secret parameter

- **Security of keys**. We need to ensure the security of chunk-level keys in the SP and avoid single-point-of-failure of file-level key. Specifically, keys are secure even if the adversaries collude with key servers (less than t).

In our threat model, we only focus on protecting the data confidentiality and key security in the SP and the DKS.

IV. Design of SecDep

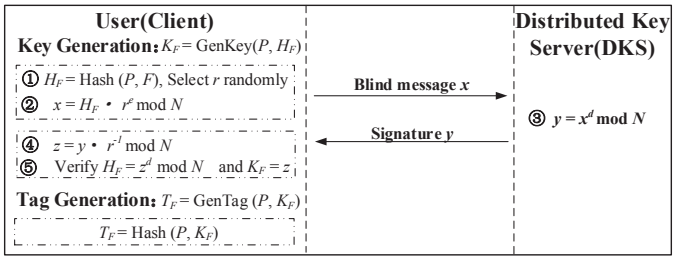
A. Overview of SecDep

In this section, we introduce our scheme, SecDep, which employs User-Aware Convergent Encryption (UACE) and Multi-Level Key management (MLK) approaches. First, UACE performs cross-user deduplication at file-level that encrypts files with the server-aided CE key (see subsection IV-B). Meanwhile, UACE performs inside-user deduplication at chunk level that encrypts chunks with the user-aided CE key (see subsection IV-B). Second, MLK encrypts chunk-level keys with the corresponding file-level key and splits file-level key into secure key shares via Shamir Secret Sharing Scheme, and sends them to the Distributed Key Servers (see subsection IV-C). Third, we will introduce some discussion and analysis on the role of the Distribute Key Servers (see subsection IV-C). Finally, we describe the workflow of the backup and restore protocols of SecDep in details (see subsection IV-D).

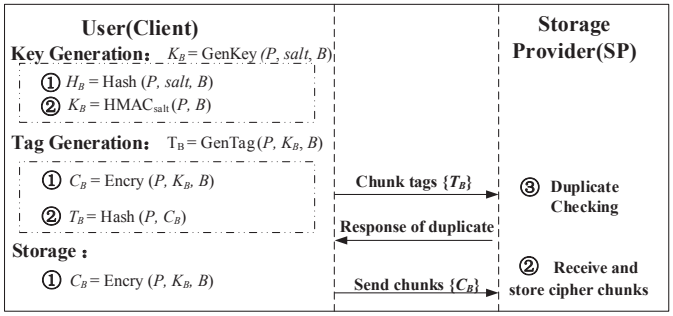
B. User-Aware Convergent Encryption

In order to resist brute-force attacks and reduce computation (time) overheads, UACE combines cross-user file-level and inside-user chunk-level secure deduplication, and exploits different secure policies for better performance.

In general, (1) UACE firstly performs cross-user file-level Hash Convergent Encryption (HCE) when each user backups their files. For each file, the user computes a file-level key and file tag via server-aided HCE. The user sends file tag to the SP and searches file tag in the global file-tag index, and then the SP returns the file deduplication results to the user. (2) If it is not a duplicate file, it will be divided into several data chunks. The CE keys and tags of these chunks will be computed by



(a) Cross-user file-level hash convergent encryption



(b) Inside-user chunk-level convergent encryption

Fig. 3. UACE: A User-Aware Convergent Encryption algorithm.

performing user-aided Convergent Encryption (CE). The user then sends the chunk tags to the SP. The SP will check whether the chunk tags are existed in the tag index of this user, and return the duplicate-checking results to the user. Then the user encrypts all the unique chunks and sends ciphertexts of chunks to the SP. To facilitate understanding, the notations used in this paper are listed in TABLE IV.

Cross-user file-level Hash Convergent Encryption (HCE): As shown in Figure 3(a), cross-user file-level hash convergent encryption mainly consists of two steps, i.e., key and tag generations. Specifically, it encrypts data by the server-aided HCE where the CE keys are added secret by a key-server via an oblivious PRF protocol. The oblivious PRF protocol can be built from deterministic blind signatures [9, 23] and we adopt RSA-OPRF scheme, which is based on RSA blind signatures [24].

- **GenKey**(P, H_F) is the key generation function. The public RSA exponent e is fixed. The key generation uses e to get (N, d) such that $e \cdot d \equiv 1 \bmod \phi(N)$, where modulus N is the product of two distinct primes of roughly equal length and $N < e$. Then the public key (N, e) and private key (N, d) are returned. For each input file F, the user chooses a random number $r \in \mathbb{N}$, gains H_F via computing hash of F and sends $x = H_F \cdot r^e \bmod N$ to the DKS. The DKS computes y using the equation $y = x^d \bmod N$ and sends y back. The user just calculates $z = y \cdot r^{-1} \bmod N$. The user could also verify whether or not $H_F \equiv z^d \bmod N$. Hence, z is the file-level key.
- **GenTag**(P, K_F) is the tag generation function of HCE that takes public parameter P and file key K_F as inputs. File tag could be implemented by T_F = Hash(P, K_F).

The main reason why CE suffers from brute-force attacks is that CE is deterministic and thus keyless [9]. We used a random secret to make the file-level key randomly to protect against brute-force attacks. In order to support cross-user deduplication, users need a key server to add a global secret to ensure the randomness of the file-level keys. As long as the server-aided CE key is kept securely, the adversaries cannot break the confidentiality of data. The message sent to DKS is blinded by the random number selected by user themselves. DKS and the adversaries cannot get the file hash even if they obtain the median values.

Inside-user chunk-level convergent encryption: The user encrypts and deduplicates chunks inside user via user-aided CE. As shown in Figure 3(b), if it is not a duplicate file, UACE will perform inside-user chunk-level deduplication where the key generations are aided by users via adding secret information. The secret information, also called "salt", is kept secure by the user. Then we will introduce key generation, tag generation, and storage of non-duplicate chunks.

- **GenKey**(P, $salt$, B) is the key generation function that takes public parameter P, a secret information $salt$ and chunk B as inputs. (1) The user calculates chunk hash value H_B = Hash(P, B). (2) The user computes the chunk-level key via K_B = HMAC$_{salt}$(P, K), and HMAC could be implemented by HMAC-SHA256. "Salt" is the secret provided by the user.
- **GenTag**(P, K_B, B) is the tag generation algorithm that takes P, K_B and B as inputs. (1) The user encrypts chunk and gets the chunk ciphertext C_B via Encry(P, K_B, B) $\longrightarrow C_B$. (2) The user gets chunk tag T_B via Hash(P, C_B) $\longrightarrow T_B$. (3) The user sends chunk tag to Storage Provider (SP) for duplicate checking. The SP searches the chunk tag in chunk-tag index and returns chunk-level deduplication results to the user.
- **Storage** of non-duplicate chunks. (1) The user encrypts the unique chunks with their CE chunk-level keys by Encry(P, K_B, B) $\longrightarrow$ C$_B$. (2) The user sends all chunk ciphertext C_B to the SP. The SP receives C_B and writes them to storage devices.

As mentioned above, UACE combines cross-user file-level and inside-user chunk-level secure deduplication, and exploits different secure polices, namely, server-aided HCE among users and user-aided CE inside users to ensure data & key security with low overheads.

Discussions: UACE provides a reasonable trade-off between dedup factor and data security. Then we discuss the reasons why UACE achieves a high dedup factor and significantly reduces time overheads without compromising data security.

(1) Cross-user redundant data are mainly from duplicate files. Hence, cross-user file-level deduplication could eliminate most of the duplicate data among multiple users. UACE combines cross-user file-level and inside-user chunk-level d-eduplication. UACE eliminates duplicate data both among and inside users, which achieves a high dedup factor.

(2) To resist brute-force attacks, cross-user and inside-user

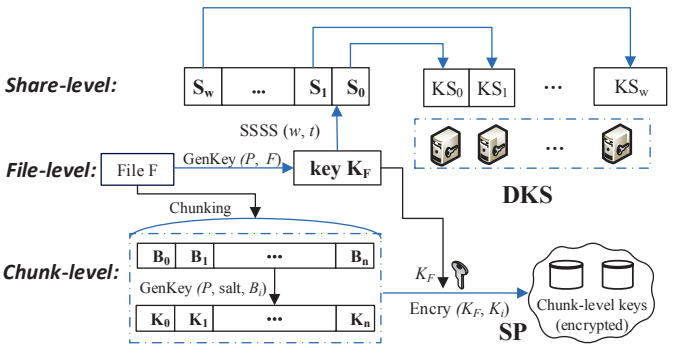


Fig. 4. MLK: A Multi-Level Key management approach for secure deduplication, which contains keys at file-level, chunk-level and share-level. SSSS (w, t) is that Shamir Secret Sharing Scheme takes w, t as inputs.

deduplication face different security challenges. Sharing keys or secret among users does not work due to user compromise. However, we add "secret" information to convergent keys to resist brute-force attacks. UACE uses server-aided HCE at file-level and user-aided CE at chunk-level deduplication to enable security and deduplication.

(3) UACE is time-efficient and reduces computation overheads significantly compared with chunk-level DupLESS approach (see Subsection VII-D). The key and tag generations of chunk-level DupLESS approach brings a large amount of computation overhead. On the contrary, UACE exploits efficient key and tag generations in both file-level and chunk-level deduplication. Specifically, UACE uses user-aided CE with lower overheads at chunk-level deduplication via adding secret by users themseleves. In addition, UACE adopts HCE at file level to generate file keys and tags, which offers better performance than CE.

C. Multi-Level Key Management

Existing convergent key management approaches have variable limitations. The main problem is that key space overhead will increase with the number of sharing users or secret shares [12]. In order to reduce key space overheads and ensure key security, we propose a Multi-Level Key management approach (MLK). As shown in Figure 4, we have a three-level hierarchy, including file-level, chunk-level, and share-level keys.

- **File-level:** For each input file F, MLK generates the file-level key K_F by GenKey(P, H_F).
- **Chunk-level:** For a non-duplicate file F, it will be divided into chunks $\{B_i\}(i = 1, 2, ...)$. Then, each chunk B_i is assigned a key K_i by GenKey(P, K_F, K_i). MLK encrypts $\{K_i\}$ with K_F by Encry(P, K_F, K_i). The ciphertext of chunk-level keys will be sent to the SP.
- **Share-level:** As for file key K_F, MLK splits it into secure share-level keys $\{S_j\}(j$ = 1, 2, ..., $w)$ by Shamir Secret Sharing Scheme (w, t). And MLK distributes $\{S_j\}$ to Distributed Key Servers (DKS) through secure channel. Each key server stores and ensures security of share-level keys.

The state-of-the-art key management approaches incur large key space overheads. For example, the key space of Master key increases linearly as a function of the number of sharing users [12]. Dekey's key space increases as a function of the number of shares. However, MLK uses file-level key to encrypt chunk-level keys so that it avoids key space overheads increasing linearly with the number of sharing users. For this reason , MLK also reduces time overheads for computing keys (see subsection VII-E).

Existing key management approaches also suffer from a single point of failure risk [12]. Specifically, if the single key server fails or the master key is compromised by adversaries, all users' data will be leaked or corrupted. To solve this problem, MLK splits file-level key into secure share-level via Shamir Secret Sharing Scheme and sends secure share-level keys to the Distributed Key Servers. MLK also backups chunk-level keys to the SP. As a result, MLK ensures key security and addresses the problem of single-point-of-failure.

Role of the DKS: . The function of the DKS are twofold, namely, aiding secret information to generate random file-level keys, and storing share-level keys. (1) To resist brute-force attacks, key servers aid the users to generate file-level keys by adding secret information via RSA-OPRF [9]. Users do not trust the DKS completely. Key servers do not know the file hash H_F and file key K_F because the user blinds the above data. (2) In order to ensure key security and avoid single-point-of-failure of file-level keys, MLK splits file-level keys into share-level keys and sends them to the DKS via a secure channel.

D. The Workflow of SecDep

System Setup. The system setup in SecDep initializes some parameters of Convergent Encryption (CE) and Shamir Secret Sharing Scheme (SSSS). The parameter 1^λ is initialized to generate a public parameter P via P = Setup(1^λ). The number of key servers is w. The parameter (w, t) of Shamir Secret Sharing Scheme is initialized with the number of key servers. The private key (N, d) and public key(N, e) are generated and securely distributed to the DKS and users. Each user has to generate a secret "salt" and keep it secure. Figure 5 describes the workflow of SecDep. We will show the backup and restore protocols as follows.

Backup protocol. The DKS has obtained RSA private key (N, d) [25]. Public key (N, e) has been published to users.

S1: For each input file F, the user generates file hash H_F = Hash(P, F) of file F. Then the user generates file key K_F = GenKey(P, H_F), which is aided by a key server. Next, the user generates the file tag T_F = GenTag(P, K_F), and sends T_F to the SP.

S2: The SP receives file tag T_F and checks it whether it exists in the file-tag index or not. If so, the SP will set IsDup = "Yes" and set a pointer to file recipe of file F. Otherwise, the SP sets IsDup = "No" and returns results to the user.

S3: The user receives the IsDup result. If IsDup = "Yes", the user only needs to update file metadata and the backup

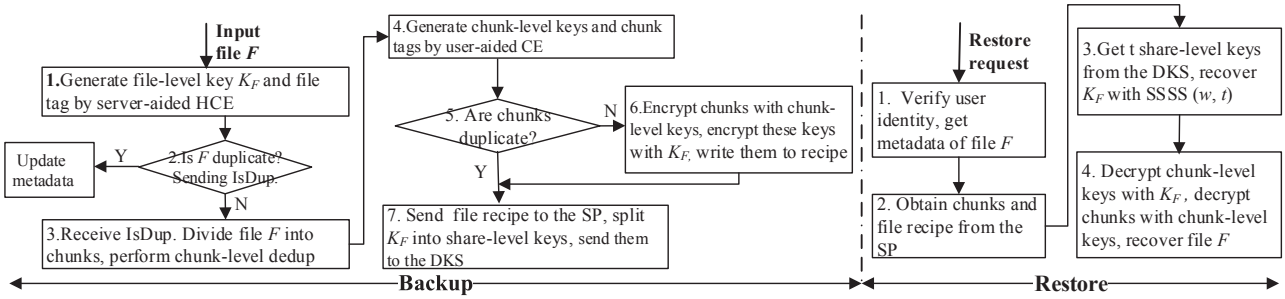


Fig. 5. The workflow of SecDep, including backup and restore protocols. SSSS(w, t) represents Shamir Secret Sharing Scheme (w, t).

process ends. If IsDup = "No", the user performs inside-user chunk-level deduplication.

S4: For chunk-level deduplication (i) the user divides F into chunks $\{B_i\}$ $(i = 1, 2, ...)$ via content-defined chunking algorithm. (ii) For each B_i, the user generates the chunk-level key K_{B_i} = GenKey(P, salt, B_i). (iii) The user uses K_{B_i} to encrypt the chunk, C_{B_i} = Encry(P, K_{B_i}, B_i). (iv) The user generates chunk tag T_{B_i} = GenTag(P, C_{B_i}) and sends T_{B_i} to the SP for checking duplicate chunks.

S5: All chunk tag $\{T_{Bi}\}$ will be sent to the SP and searched on chunk-tag index. For each chunk B_i $(i = 1, 2, ...)$, if it is a duplicate chunk, the SP sets ψ[i] = 1. Otherwise, sets ψ[i] = 0.

S6: If ψ[i] = 0, the user uploads the ciphertext of chunk C_{B_i} to the SP. And the user needs to create a file recipe for efficiently restoring the file after deduplication. The user uses file-level key K_F to encrypt chunk-level keys $C_{K_{B_i}}$ = Encry(P, K_F, K_{B_i}). Meantime, the user also writes the corresponding ciphertext of chunk-level keys to the file recipe and sends the file recipe to the SP.

S7: If ψ[i] = 1, the user updates file recipe. The user splits file-level key K_F into w different share-level keys $\{S_j\}$ by SSSS (w, t) [19] and distributes them to the DKS. The DKS receives and stores share-level keys $\{S_j\}$ $(j = 1, 2, ..., w)$.

Restore protocol. When the user wants to restore a file, the user has to send a request and file name to the SP.

S1: The SP receives the restore request and verifies the user's identity. If it fails, the SP rejects user's request.

S2: If it passes, the SP will read and send all these corresponding chunks and file recipe to the user. The user receives file recipe and all ciphertext of chunks.

S3: The user sends the restore request to t different key servers randomly for obtaining share-level keys. Each key server verifies the identity of the user. And the key servers read the share-level keys $\{S_j\}$ and send them to user via a secure channel. The user receives key shares and combines them to recover file key K_F via SSSS(w, t).

S4: The user gets the file recipe and file-level key. The user decrypts and obtains each chunk-level key K_{B_i} = Decry(P, K_F, $C_{K_{B_i}}$). For each chunk C_{B_i} (i=1, 2, ...), the user decrypts chunks B_i = Decry(P, K_{B_i}, C_{B_i}). Finally, the file F is recovered.

V. Implementation Details

We have implemented our prototype SecDep based on User-Aware Convergent Encryption (UACE) and Multi-Level Key management (MLK) approaches. As shown in Figure 2 in Section III, SecDep consists of client, the Storage Provider (SP), and Distributed Key Servers (DKS). They are used as the three key entities of our system model. The client takes user's identity credentials and requests as inputs. We use hash and encryption functions based on OpenSSL library, for example, SHA-256 and AES-256 [22], also used in DupLESS [9] and Dekey [12]. SecDep performs cross-user file-level and inside-user chunk-level deduplication. Client implements the following functions,namely, chunking algorithm, file-/chunk-level key/tag generation, and encryption. Our Storage Provider (SP) maintains file-tag and chunk-tag indices for duplicate checking. The SP stores file recipe, chunks, and chunk-level keys. Each key server in the DKS can receive and store key shares. We will describe the chunking, index management, and container management in details as follows.

A. Chunking

We implement both cross-user file-level and inside-user chunk-level deduplication. As for chunk-level deduplication, we implement both Fix-Sized and Content-Defined Chunking algorithms, where Content-Defined Chunking (CDC) [26] is default. The average chunk size are configured to 2KB, 4KB, and 8KB respectively, which will be evaluated in Section VII.

B. Index Management

In order to perform deduplication, the SP has to maintain the indices of file and chunk tags. The user computes the tags of files and chunks, and uploads them to the SP for duplicate-checking. Specifically, SecDep rents a co-locating Virtual Machine (VM) and a cloud storage backend to support duplicate checking and data storage [27]. To support efficient cross-user file-level deduplication and inside-user chunk-level deduplication, SecDep manages indices of file and chunk tags by employing the wildly used locality-based deduplication indexing scheme [28, 29].

First, the file-tag index holds file tag entries for different users. Each entry of a file is identified by the file name and tag. The entry stores a reference to the file recipe, which consists of tags and sizes of the chunks in sequence. Second,

the chunk-tag index holds chunk tag entries inside users. Each chunk entry is identified by its tag and keeps a reference to the container that stores the physical chunk. SecDep does not have to maintain a global chunk tag index among users but just several individual inside-user chunk tag indices.

C. Container Management & Storage Protocol

During a backup, the chunks that need to be written are aggregated into containers to preserve the locality of the backup stream. The default container size is 4MB. SecDep supports backup and restore protocols that are detailed in subsection IV-D. After users delete expired backups, chunks become invalid (not referenced by any backup) and must be reclaimed. The simplest reference counting technique [30] is used to support deletion and garbage collection in SecDep. The more sophisticated garbage collection schemes will be considered into SecDep as our future work [31].

VI. Security Discussion

SecDep is designed to ensure data confidentiality and key security for cross-user fine-grained deduplication-based system for cloud backups. In SecDep, we consider two types of adversaries, that is, external adversary and internal adversary. However, we only focus on internal adversary because SecDep could resist the external attacks by authentication [12]. We assume that the following technologies are secure, such as Shamir Secret Sharing Scheme [19] and symmetric encryption. We analyze data confidentiality and security of keys in the case the adversary compromises the SP, colludes with users, or steals data from key servers. Thus we present security analysis of SecDep in a multi-tiered way.

A. Security of Data

In the case that the adversary tries to compromise the SP or collude with users, SecDep could resist brute-force attacks and duplicate-faking attacks to ensure data security, including the confidentiality and integrity.

The adversary tries to obtain the content of files from other legitimate users. The adversary may compromise the SP to get the chunks on the SP and perform brute-force attacks. Specifically, the adversary obtains the ciphertexts of target chunks from a specific file. The adversary knows that the chunks are from a specific set $|S|$. For each chunk, the adversary encrypts it to get the ciphertext and compares it with the target chunk. Then the adversary gets the original file. However, SecDep can still ensure data confidentiality. All users' data that are uploaded to the SP, have been encrypted with chunk-level keys. The chunk-level keys are generated by adding secret via user-aided CE. In general, it is difficult to break the confidentiality of users' data because the adversary doesn't know the secret.

The adversary colludes with some users and performs duplicate-faking attacks [18, 20] to the data on the SP, which compromises the integrity of users' data. Specifically, the adversary and these colluders may upload their data to the deduplication-based system for cloud backups. They upload the correct tags, but replace the chunks with the wrong data. To address this problem, SecDep can compute the hash value of the ciphertext of chunks. Then the user compares hash value with its' tag to resist duplicate-faking attacks.

TABLE VI
Workload characteristics of four real-world datasets.

Characteristics	One-set	Inc-set	Full-set	FSLhomes
Number of users	11	6	19	7
Total size	491GB	224.4GB	2.5TB	14.5TB
Total files	2.5M	0.59M	11.3M	64.6M
Total chunks	50.5M	29.4M	417M	1703.3M
Avg. chunk size	10KB	8KB	6.5KB	8KB
Dedup factor	1.7	2.7	25	38.6

B. Security of Keys

The adversary tries to obtain the keys, and recover other users' data. Specifically, (1) the adversary gets chunk-level keys by compromising the SP. However, chunk-level keys are encrypted by file-level key via symmetric encryption. SecDep can ensure the security of chunk-level keys as long as file-level keys are stored securely. (2) The adversary tries to get share-level keys from key servers and recover the file-level key, which is very difficult. This is because the share-level keys in SecDep are securely distributed in several key servers by using the known Shamir Secret Sharing Scheme (SSSS) [19].

C. Security of SecDep

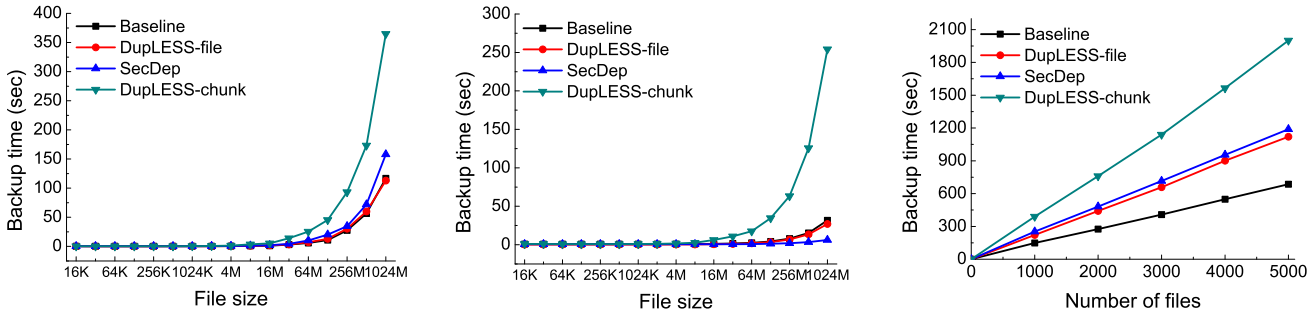
As mentioned above, the adversary can't obtain other users' data if it only compromises the SP, colludes with users or steals data from key servers. Then we discuss the security of SecDep if the adversary tries to compromise the SP or collude with users.

In the best case, the adversary compromises the SP, but cannot access to the DKS. All data and metadata stored in the SP is encrypted with random keys, including file recipe. The adversary cannot know the content of other users' data even if it performs brute-force attacks. **In the semi-best case**, the adversary has compromised some users and have been authorized access to DKS. SecDep can still ensure data security. Although adversary can perform brute-force attacks on chunks via CE, he/she can't break the encryption key due to not knowing the user's secret. **In the worst case**, if the adversary has obtained some users' secret and other users' ciphertexts of chunks, SecDep can still ensure security for unpredictable data that are not falling into a known set. For the worst case that adversary attacks the predictable data within a known set, SecDep makes the worst case rarely occur by further protecting tags and file metadata [9].

VII. Performance evaluation

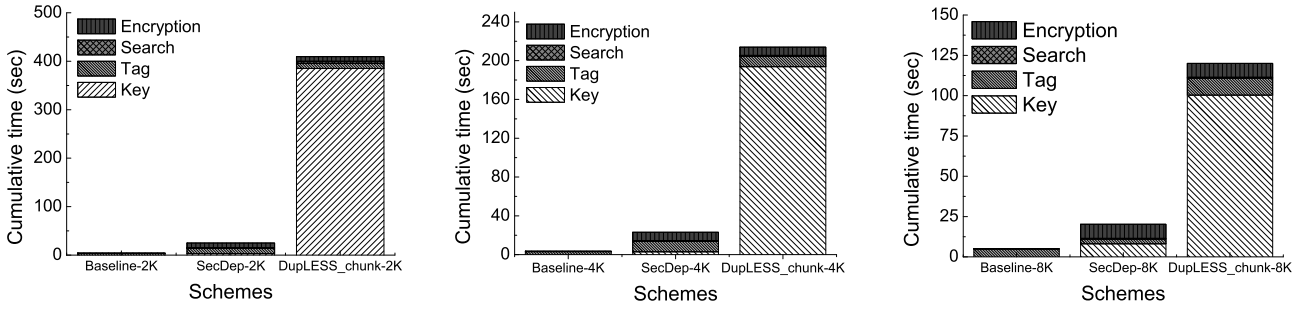
A. Experimental Setup

In order to evaluate the performance of SecDep, We conduct the experiments using machines equipped with an Intel(R) Xeon(R) E5606@2.13GHZ 8 Core CPU, 16GB RAM, and installed with Ubuntu 12.04 LTS 64-bit Operation System.



(a) Different file size with non duplicates (b) Different file size with full duplicates (c) Number of files

Fig. 6. The upload time comparison among Baseline, DupLESS-file, SecDep, and DupLESS-chunk on different sizes and numbers of files.



(a) Average chunk size of 2KB (b) Average chunk size of 4KB (c) Average chunk size of 8KB

Fig. 7. The time breakdown comparison among Baseline, SecDep, and DupLESS-chunk with different average chunk sizes.

These machines are connected with 100Mbps Ethernet network. We implement a research prototype to evaluate and compare the performance of different schemes, including Baseline, DupLESS-file, DupLESS-chunk, and SecDep. Baseline is a basic chunk-level deduplication system without any security mechanisms. DupLESS-file and DupLESS-chunk are implementing secure DupLESS [9] schemes at file and chunk levels respectively. Note that our evaluation platform is not a production quality secure deduplication system but rather a research prototype. Hence, our evaluation results should be interpreted as an approximate and comparative assessment of other secure deduplication approaches above, and not be used for absolute comparisons with other deduplication systems.

We use both of the synthetic datasets and real-world datasets for evaluation. The synthetic datasets consist of artificial files filled with random contents and each file is divided into fixed-sized chunks. Real-world datasets are summarized in Table VI, including One-set, Inc-set, Full-set and FSLhomes. One-set was collected from 11 graduate students of a research group and was reported by Xia et al. [32]. Inc-set was collected from initial full backups and subsequent incremental backups of 6 members of a university research group and was reported by Tan et al. [21]. Full-set consists of 380 full backups of 19 researchers' PC and is reported by Xing et al. [33]. FSLhomes is a public dataset reported by Tarasov et al. [34] and can be downloaded from website [35]. FSLhomes contains snapshots of students' home directories, where files consist of source code, binaries, office documents, and virtual machine images.

In order to compare the performance of the existing secure deduplication schemes and SecDep, we mainly use dedup factor, backup time, key space and key protecting time as the quantitative metrics. Dedup factor is defined as the ratio of the data sizes before/after deduplication. Backup time consists of key and tag generations, searching, encryption, and transferring time. Key space is the key storage overheads. Key protecting time is the encryption or encoding time for key management. We observe the impacts of varying sizes & numbers of files and average chunk sizes on the system performance.

B. A Sensitivity Study on Size & Number of Files

In this section, we evaluate the impacts of varying file size on performances of Baseline, DupLESS-file, SecDep, and DupLESS-chunk. Files are generated with random contents of size 2^i KB for i $\in \{4, 5, ..., 20\}$, giving us a file-size ranging of from 16 KB to 1024 MB. To evaluate the impacts of varying the number of files on backup time, we upload 1000-5000 1MB unique files filled with random contents.

Figure 6(a) shows the results that SecDep significantly reduces backup time compared with DupLESS-chunk as discussed in subsection IV-B. When file size is small (i.e., 16KB-4MB), backup time of these schemes is similar. It is because key and tag generations account for little part of the whole time. Starting from 8MB, DupLESS-chunk incurs significantly higher overheads than others. This is because of modular exponentiation in RSA-OPRF protocol, which will incur huge

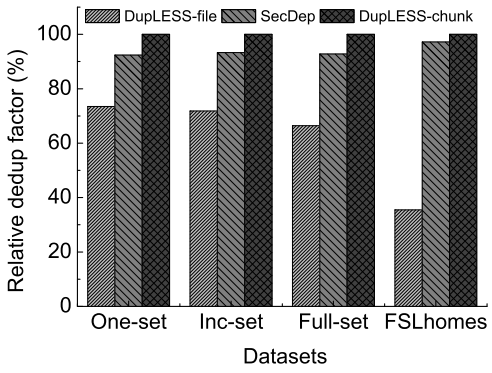


Fig. 8. The comparisons of DupLESS-file, SecDep, and DupLESS-chunk in terms of relative dedup factor. The Y-axis shows the relative dedup factor to DupLESS-chunk.

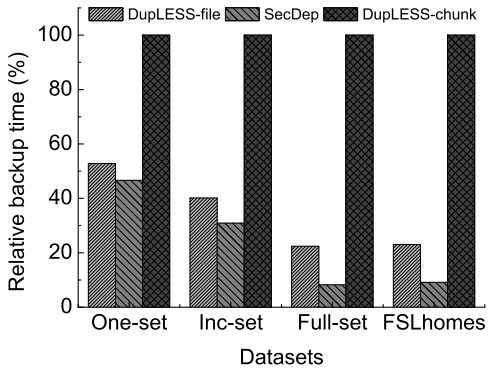


Fig. 9. The backup time overheads of DupLESS-File, SecDep, and DupLESS-Chunk on four real-world datasets. The Y-axis shows the relative backup time to dupless-chunk.

TABLE VII
THE KEY SPACE OVERHEAD COMPARISON AMONG THE STATE-OF-THE-ART CHUNK-LEVEL KEY MANAGEMENT APPROACHES.

Approach	Key space overheads
Single Key Server	$O = \frac{D*S}{k*DF}$
Master Key	$O = \frac{D*S}{k}$
SecDep	$O = \frac{D*S}{k*DF} + N_u*w*S$
Dekey	$O = \frac{D*S}{k*DF} * \frac{w}{t-r}$

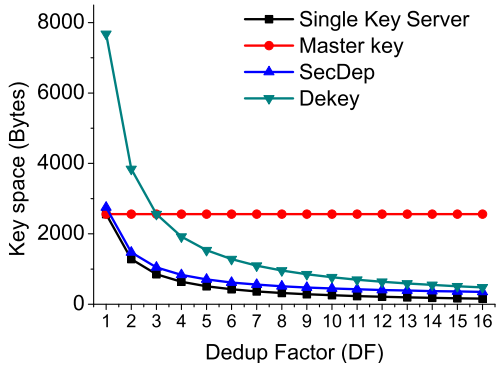


Fig. 10. The key space overhead per file of Single Key Server, SecDep, Master Key, and Dekey under dedup factor.

computation and time overheads for chunk-level deduplication.

As shown in Figure 6(b), SecDep incurs less time overheads compared with DupLESS-chunk when all files are duplicate. It is because SecDep only performs file-level deduplication among users, which avoids the time-consuming chunk-level RSA-OPRF protocol in DupLESS-chunk. We also observe that SecDep is slightly better than DupLESS-file. It is because SecDep uses the server-aided HCE in to reduce the encryption overhead while DupLESS uses the conventional CE.

In Figure 6(c), backup time increases linearly with the number of files. SecDep is more time-efficient than DupLESS-chunk regardless of the number of files.

C. A Sensitivity Study on Average Chunk Size

To evaluate the time overhead of some individual step under different chunk size, we upload a 512 MB unique file repeatedly but with different chunk sizes, i.e., 2KB, 4KB, and 8KB. The evaluated steps consist of key & tag generations, index searching, and encryption. Note that data transfer is not included.

Figure 7(a), 7(b), and 7(c) shows that the cumulative time is inversely proportional to the average chunk size. And DupLESS-chunk incurs large time overheads than SecDep. Figure 7(a) shows that tag generation and key generation account for 96.5% time overheads. RSA-OPRF is time-consuming, which is used to generate the server-aided keys by DupLESS-chunk. Large number of chunks incur huge computation overheads as discussed in Section II.

In Figure 7(a)-7(c), SecDep consumes 90-96.5% less time overheads for key and tag generations than DupLESS-chunk. SecDep uses UACE, which employs server-aided HCE on file-level and user-aided CE on chunk-level deduplication to balance the deduplication performance and data security.

D. Evaluating SepDep on the Four Real-world datasets

In Figure 8, we evaluate the relative dedup factor of the four real-world datasets. The normalized dedup factor of DupLESS-chunk is regarded as 100%, while the real dedup factor is listed in Table VI. We evaluate the relative backup time of DupLESS-file, SecDep, and DupLESS-chunk under these datasets. The backup time of DupLESS-chunk is regarded as 100%. We measure and record the average time of each individual step, for example, tag and key generations at file-level and chunk-level, index searching, encryption, and data transfer. Based on these time, we obtain the backup time of four real-world datasets.

As shown in Figure 8, SecDep eliminates the majority of duplicate data, only resulting in a 2.8-7.35% loss of dedup factor compared with the DupLESS-chunk as discussed in subsection II-C. It is because SecDep combines cross-user file-level and inside-user chunk-level deduplication, which efficiently eliminates most of duplicate data in backup datasets.

Figure 9 suggests that SecDep reduces 52-92% of backup

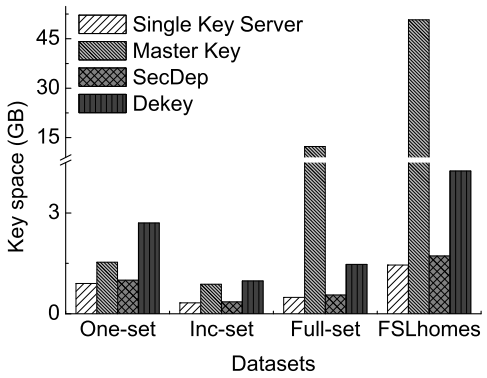


Fig. 11. The key space overheads consumed by the four key management approaches on the four real-world datasets.

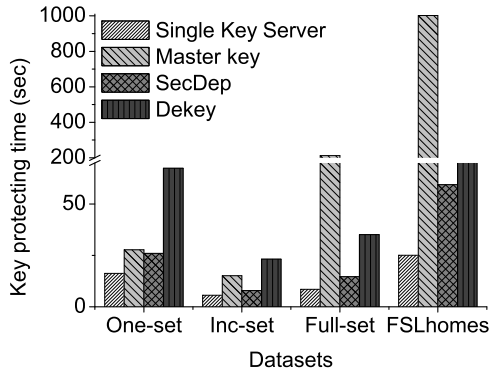


Fig. 12. The key protecting (i.e., encrypting or encoding) time overheads consumed by the four key management approaches on four real-world datasets.

time overheads compared with DupLESS-chunk as discussed in Section II-C and IV-B. There are two reasons. i) When files are duplicates, SecDep just performs file-level deduplication via server-aided HCE, which offers better performance than DupLESS-file using the conventional CE. ii) SecDep uses user-aided CE inside users to generate keys, which avoids the time-consuming RSA-OPRF operations of DupLESS-chunk.

Figure 9 shows that SecDep is more time-efficient than DupLESS-file. When files are duplicate, SecDep reduces much more time overheads on key and tag generations compared with DupLESS-file because HCE offers better system throughput than CE. SecDep reduces more transfer time compared with DupLESS-file due to the higher dedup factor.

E. Space & Computation Overheads for Key Management

In this subsection, we evaluate the space and computation overheads of SecDep's Multi-Level Key management. First, we describe the key space overheads (O) of different secure fine-grained deduplication approaches in Table VII. Then we evaluate the impacts of varying dedup factor on key space overheads per file in Figure 10. Finally, we evaluate the space and computation overheads for key management on the four real-world datasets in Figures 11 and 12 respectively. The parameters used in Table VII are defined as follow. D is data size, N_u is the number of unique file, k is the average chunk size, S is the size of chunk key, and DF is the dedup factor. RSSS(w, t, r) [12] and SSSS(w, t) [19] take w, t, and r as parameters, which are set to 6, 4, and 2 respectively.

In Table VII, the key space of Dekey are $w/(t-r)$ times of the number of chunk-level keys [12]. SecDep adds $N_u * w * S$ shares over the Single Key server approach, which only accounts a very small fraction of the total key space overhead. As shown in Figure 10, the key space overheads of SecDep will be decreased with the deduplication factor and SecDep adds little key space overheads compared with Single Key Server approach. Note that the Master Key approach has the highest key space overheads, which is because that it always encrypts chunk-level keys with user' own master key regardless of whether the files are duplicates or not.

Figure 11 suggests that SecDep reduces 59.5-63.6% and 34.8-96.6% of key space overheads on the four reel-world datasets compared with Dekey and Master Key approach respectively. Because SecDep uses MLK, which employs file-level key to manage chunk-level keys to avoid the key space overheads increasing with the number of users as discussed in subsection IV-C.

Figure 12 shows the encryption/encoding time overheads of different key management approaches on the four real-world datasets. The Master Key approach incurs large time overheads while SecDep is time-efficient because SecDep reduces the amount of encrypting chunk-level keys with using file-level keys. The results of decrypting and decoding time overheads are similar to that shown in Figure 12.

In summary, SecDep significantly reduces backup time overheads compared with DupLESS-chunk approach based on four real-world datasets, regardless of the variety of size, number, and average chunk size. Meanwhile, SecDep also reduces key space overheads and is more time-efficient on key management compared with Dekey and Master Key approaches.

VIII. Related Work

Data deduplication techniques could be classified into two types based on the granularity: file-level and chunk-level. Many papers [2, 36] present performance comparison between file-level and chunk-level deduplication methods, which suggests cross-user redundant data are mainly from duplicate files. Tan et al. [21] propose SAM, which combines the global file-level deduplication and local chunk-level deduplication, which makes a trade-off between the deduplication efficiency and indexing overheads.

Recently, data confidentiality for deduplication-based cloud storage systems is gaining increasing attention [37]. Douceur et al. [10] propose and formalize convergent encryption to enable deduplication and encryption, which uses its content hash as a key to encrypt data via a deterministic encryption scheme. Storer et al. [20] adopt convergent encryption and explore secure data deduplication based on the authenticated and anonymous model. Bellare et al. [18] put forward that

convergent encryption is subject to brute-force attacks because of deterministic and keyless problems. They formalize this primitive as message-locked encryption [18] and develop DupLESS [9] system which uses convergent encryption for file-level deduplication storage while keys are obtained from a key server via RSA-OPRF protocol [9]. Stanek et al. [38] present a scheme that guarantees different security level for popular and unpopular data.

Meanwhile, key management for secure deduplication is also gaining attention recently. Generally, customers have to backup their convergent keys for the security concerns. Hence, fine-grained key management approach becomes a challenge due to large number of chunks. Storer et al. [20] and ClouDedup [13] propose that they will send and store convergent keys to a key server. However, DupLESS requires that each user uses a *Master Key* to encrypt convergent keys and sends ciphertext of keys to cloud storage. Li et al. propose Dekey [12] that splits chunk-level key into key shares via RSSS [14, 15]. However, they do not discuss key space overheads and evaluate their approach with the real-world datasets.

To resist attacks that access files based on a small hash value, Harnik et al. [39] introduce "proofs of ownership" (PoWs) for deduplication system. PoWs is a protocol using Merkel Hash Tree and erasure code that lets users prove to a server whether or not they hold the file. Xu et al. [40] propose a secure client-side deduplication scheme with efficient min-entropy in a bounded leakage setting. Pietro et al. [41] introduce s-PoW, an efficient and secure PoW scheme that outputs a proof with each bit selected at a random position of the file. Ng et al. [42] presents a new PoW scheme for encrypted data on private data deduplication, but they do not consider the key space overheads.

IX. Conclusion

In this paper, we study the problems of data confidentiality and key security for cross-user fine-grained deduplication in cloud backup systems. We observe that state-of-the-art secure deduplication methods either suffer from brute-force attacks, or incur large computation (time) overheads, and key management approaches incur large key space overheads. We design SecDep, which exploits redundant data distribution on cross-user file-level and inside-user chunk-level to perform different security policies, to make a trade-off between the data security and deduplication performance.

In general, SecDep mainly consists of two modules: User-Aware Convergent Encryption algorithm (UACE) and Multi-Level Key management (MLK) approaches. (1) UACE uses server-aided HCE at file-level and user-aided CE at chunk-level to resist brute-force attacks. Moreover, UACE uses an efficient user-aided CE approach at fine-grained deduplication to reduce computation overheads. Four real-world datasets driven experimental results show that UACE reduces 52-92% of time overheads at the expense of only losing 2.8-7.35% of dedup factor compared with the state-of-the-art DupLESS approach. (2) MLK encrypts chunk-level keys by file-level key, which avoids key space increasing with the number of sharing users. Meanwhile, MLK splits file-level key into share-level keys via Shamir Secret Sharing Scheme and sends them to the Distributed Key Servers, which ensures key security and avoids single-point-of-failure. Our evaluation results suggest that MLK reduces 59.5-63.6% and 34.8-96.6% of key space overheads than the state-of-the-art Dekey and Master key approaches, respectively.

Acknowledgments

This work was partly supported by the National Basic Research 973 Program of China under Grant No.2011CB302301; NSFC No. 61025008, 61173043, 61232004, and 6140050892; 863 Project 2013AA013203; Fundamental Research Funds for the Central Universities, HUST, under Grant No. 2014QNRC019; Director Fund of WNLO. This work was also supported by Key Laboratory of Information Storage System, Ministry of Education, China. The authors are also grateful to anonymous reviewers and our shepherd, James Hughes, for their feedback and guidance.

References

[1] "The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things," http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm, April 2014, EMC Digital Universe with Research & Analysis by IDC.

[2] D. Meyer and W. Bolosky, "A study of practical deduplication," in *Proceedings of the USENIX Conference on File and Storage Technologies*. San Jose, CA, USA: USENIX Association, February 2011, pp. 229–241.

[3] G. Wallace, F. Douglis, H. Qian, and et al, "Characteristics of backup workloads in production systems," in *Proceedings of the Tenth USENIX Conference on File and Storage Technologies (FAST'12)*. San Jose, CA: USENIX Association, February 2012, pp. 1–14.

[4] A. El-Shimi, R. Kalach, A. Kumar, and et al, "Primary data deduplication-large scale study and system design," in *Proceedings of the 2012 conference on USENIX Annual Technical Conference*. Boston, MA, USA: USENIX Association, June 2012, pp. 1–12.

[5] "Dropbox," http://www.dropbox.com/, 2015.

[6] "Spideroak," https://www.spideroak.com/, 2015.

[7] "Mozy," http://www.mozy.com/, 2015.

[8] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage." in *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST'02)*. erkeley, CA, USA: USENIX Association, January 2002, pp. 89–101.

[9] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Dupless: server-aided encryption for deduplicated storage," in *Proceedings of the 22nd USENIX Security Symposium*. Washington, DC, USA: USENIX Association, August 2013, pp. 1–16.

[10] J. R. Douceur, A. Adya, W. J. Bolosky, and et al, "Reclaiming space from duplicate files in a serverless distributed file system," in *Proceedings of the 22nd International Conference on Distributed Computing Systems*. Vienna, Austria: IEEE Computer Society Press, July 2002, pp. 617–624.

[11] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," *Journal of the ACM (JACM)*, vol. 51, no. 2, pp. 231–262, 2004.

[12] J. Li, X. Chen, M. Li, J. Li, P. P. Lee, and W. Lou, "Secure deduplication with efficient and reliable convergent key management," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 6, pp. 1615–1625, 2014.

[13] P. Puzio, R. Molva, M. Onen, and S. Loureiro, "Cloudedup: secure deduplication with encrypted data for cloud storage," in *Proceedings of the 5th International Conference on Cloud Computing Technology and Science*. Bristol, UK: IEEE Computer Society Press, December 2013, pp. 363–370.

[14] G. R. Blakley and C. Meadows, "Security of ramp schemes," in *Proceedings of the 1985 Advances in Cryptology (CRYPTO'84)*. Heidelberg, Berlin: Springer, 1985, pp. 242–268.

[15] A. De Santis and B. Masucci, "Multiple ramp schemes," *Information Theory, IEEE Transactions on*, vol. 45, no. 5, pp. 1720–1728, July 1999.

[16] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: Making backup cheap and easy," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 285–298, 2002.

[17] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: the least-authority filesystem," in *Proceedings of the 4th ACM international workshop on Storage security and survivability*. Alexandria, VA, USA: ACM Association, October 2008, pp. 21–26.

[18] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Proceedings of Advances in Cryptology–EUROCRYPT 2013*. Athens: Springer, May 2013, pp. 296–312.

[19] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[20] M. W. Storer, K. Greenan, D. D. Long, and E. L. Miller, "Secure data deduplication," in *Proceedings of the 4th ACM international workshop on Storage Security and Survivability*. Alexandria, Virginia, USA: ACM Association, October 2008, pp. 1–10.

[21] Y. Tan, H. Jiang, D. Feng, and et al, "SAM: A Semantic-Aware Multi-Tiered Source De-duplication Framework for Cloud Backup," in *Proceedings of the 39th International Conference on Parallel Processing (ICPP'10)*. San Diego, CA, USA: IEEE Computer Society Press, September 2010, pp. 614–623.

[22] "Openssl project," https://www.openssl.org/, 2015.

[23] J. Camenisch, G. Neven *et al.*, "Simulatable adaptive oblivious transfer," in *Proceedings of the 27th Advances in Cryptology-EUROCRYPT*. Santa Barbara, CA, USA: Springer, August 2007, pp. 573–590.

[24] D. Chaum, "Blind signatures for untraceable payments," in *Proceedings of the 1983 ACM Advances in cryptology*. Springer, 1983, pp. 199–203.

[25] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[26] M. O. Rabin, *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[27] M. Li, C. Qin, P. P. Lee, and J. Li, "Convergent dispersal: toward storage-efficient security in a cloud-of-clouds," in *Proceedings of the 6th USENIX conference on Hot Topics in Storage and File Systems (HotStorage'14)*. Philadelphia, PA, USA: USENIX Association, June 2014, pp. 1–1.

[28] B. Zhu, K. Li, and R. H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system." in *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST'08)*. San Jose, CA, USA: USENIX Association, February 2008, pp. 1–14.

[29] W. Xia, H. Jiang, D. Feng, and Y. Hua, "Similarity and locality based indexing for high performance data deduplication," *IEEE Transactions on Computers, 2015*, vol. 64, no. 4, pp. 1162–1176, 2015.

[30] P. Strzelczak, E. Adamczyk, U. Herman-Izycka, J. Sakowicz, L. Slusarczyk, J. Wrona, and C. Dubnicki, "Concurrent deletion in a distributed content-addressable storage system with global deduplication." in *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST'13)*. San Jose, CA, USA: USENIX Association, February 2013, pp. 161–174.

[31] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, W. Xia, F. Huang, and Q. Liu, "Accelerating restore and garbage collection in deduplication-based backup systems via exploiting historical information," in *Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC'14)*. Philadelphia, PA, USA: USENIX Association, June 2014, pp. 181–192.

[32] W. Xia, H. Jiang, D. Feng, and Y. Hua, "Silo: a similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput," in *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*. Portland, OR, USA: USENIX Association, June 2011, pp. 285–298.

[33] Y. Xing, Z. Li, and Y. Dai, "Peerdedupe: Insights into the peer-assisted sampling deduplication," in *Proceedings of the 2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*. Delft, Netherlands: IEEE Computer Society Press, August 2010, pp. 1–10.

[34] V. Tarasov, A. Mudrankit, W. Buik, P. Shilane, G. Kuenning, and E. Zadok, "Generating realistic datasets for deduplication analysis." in *Proceedings of the 2012 USENIX Annual Technical Conference (ATC'12)*. Boston, MA, USA: USENIX Association, June 2012, pp. 261–272.

[35] "Fsl traces and snapshots public archive," http://tracer.filesystems.org/traces/fslhomes/2014/, 2014.

[36] C. Policroniades and I. Pratt, "Alternatives for detecting redundancy in storage systems data." in *Proceedings of the 2004 USENIX Annual Technical Conference (ATC'04)*. Boston, MA, USA: USENIX Association, June 2004, pp. 73–86.

[37] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, and E. Weippl, "Dark clouds on the horizon: Using cloud storage as attack vector and online slack space." in *Proceedings of the 20th USENIX Security Symposium*. San Francisco, CA, USA: USENIX Association, August 2011, pp. 1–11.

[38] J. Stanek, A. Sorniotti, E. Androulaki, and L. Kencl, "A secure data deduplication scheme for cloud storage," Technical Report, Tech. Rep., 2013.

[39] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *Proceedings of the 18th ACM conference on Computer and communications security (CCS'11)*. Chicago, Illinois, USA: ACM Association, October 2011, pp. 491–500.

[40] J. Xu, E.-C. Chang, and J. Zhou, "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*. Hangzhou, China: ACM Association, May 2013, pp. 195–206.

[41] R. Di Pietro and A. Sorniotti, "Boosting efficiency and security in proof of ownership for deduplication," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*. Seoul, Republic of Korea: ACM Association, May 2012, pp. 81–82.

[42] W. K. Ng, Y. Wen, and H. Zhu, "Private data deduplication protocols in cloud storage," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. Riva del Garda (Trento), Italy: ACM Association, March 2012, pp. 441–446.