Tapping the Potential: Secure Chunk-based Deduplication of Encrypted Data for Cloud Backup

Wenhai Sun, Ning Zhang, Wenjing Lou, and Y. Thomas Hou
Virginia Polytechnic Institute and State University, Blacksburg, VA, USA

***Abstract*—We, in this work, investigate the problem of designing a secure chunk-based deduplication scheme in the enterprise backup storage setting. Most of the existing works focus on realizing file-level encrypted data deduplication or key/metadata management. Little attention is drawn to the practical chunk-level deduplication system. In particular, we identify that the information contained in a small-sized chunk is more susceptible to the brute-force attack compared with file-based deduplication. We propose a *randomized oblivious key generation* mechanism based on the inner workings of the backup service. In contrast with the current work that compromising one client will eventually expose all the clients' storage, our scheme offers a counter-intuitive property of achieving *security against multi-client compromise* with *minimal deduplication performance loss*. In addition, we enforce a *per-backup* rate-limiting policy to slow down the online brute-force attack. We show that the proposed scheme is provably secure in the malicious model. We also calibrate the system design by taking into account the practical deduplication requirements to accomplish a comparable plaintext deduplication performance. Our experiment on the real-world dataset shows its efficiency, effectiveness, and practicality.**

I. Introduction

Data deduplication (or dedupe for short) is increasingly adopted by cloud storage providers, such as Amazon S3, as an effective technique to reduce the storage cost. When the system detects data redundancy, dedupe process will retain only one copy of the same data and make a reference pointing to the stored copy for other duplicates. Data confidentiality is realized by exploiting deterministic encryption, e.g., convergent encryption (CE) [1], in which the key is generated from the data itself and the same plaintext will always yield the same key and ciphertext. As a result, we can apply dedupe to the ciphertext without leaking underlying stored information. However, the existing secure deduplication designs [1], [2], [3], [4], [5], [6], to some extent, are at odds with the real-world dedupe requirements in terms of security and performance.

Knowing the Gap. By using different chunking methods, deduplication can be carried out either in the *coarse-grained* file level, or *fine-grained* chunk level. In a nutshell, file-based deduplication (FBD) is suitable for *small* or *stationary* file types, e.g., dll, lib, pdb, etc. Even small changes made in the file will lead to a completely different copy, thereby resulting in a low dedupe performance. In contrast, chunk-based deduplication (CBD) is capable of dividing a given data stream into smaller chunks of fixed or variable lengths (typically from 4KB to 16KB). As such, a considerable storage saving can be reaped from chunk-level redundancy elimination. In fact, there is a trend towards large files being the principal consumer of the storage [10]. More and more real-world storage systems are using CBD as their core deduplication technique [11]. Most of the existing works focus on secure file-level dedupe. Chunk-level designs [2], [22], [31] with different research concentration paid little attention to the challenges and practical requirements of CBD.

1) Low-entropy chunks. Deterministic CE is inherently vulnerable to brute-force attack for predictable files. For example, given the ciphertext, the adversary is able to enumerate all the file candidates, encrypt and compare them with the target ciphertext in an *offline* manner. Prior works provide solutions by deriving the encryption key from an *online* third party, i.e., either an independent key server [4] or other peer clients [6], instead of offline key generation from data itself. However, an adversary compromising one authorized client and observing the dedupe process can still launch an *online* brute-force attack. In general, the data leakage covers the storage of all clients in the system. A rate-limiting strategy may be enforced to slow down the attack speed. But such approach is merely effective if the deduplicated data has enough unpredictability. CBD will amplify the attack efficacy due to the potentially much lower entropy contained in a small chunk. Albeit it is an open problem to entirely prevent the brute-force attack, we still need to answer the question: *To what extent, can we reduce the risk of the information leakage with minimal impact on the underlying deduplication routine?*

2) Increased system operation overhead. Besides the inevitable cost of performing file chunking, directly applying existing schemes to chunk-level deduplication usually incurs higher latency and computation overhead. This is because the client needs to run the key generation protocol with other online parties (a key server [4] or peer clients [6]) to produce a CE key for each chunk of a file instead of one protocol execution for the whole file. Thus, a natural question is: *Can we speed up the key generation while still ensuring an effective deduplication function?*

3) Practical dedupe performance. In addition to the *deduplication ratio* (or *space reduction percentage*, see Sect. II) that is widely used in measuring the effectiveness of deduplication [12], there are also other metrics in practice to determine the dedupe system performance, such as *chunk fragmentation level*, or *data restore speed* (see Sect. II). Chunk fragmentation is caused by CBD in which data logically belonging to a recent backup scattered across multiple older backups [13]. A higher chunk fragmentation level typically adversely affects the system read performance and further increase the data

restore cost. On the contrary, fragmentation is not widespread in file-based deduplication owing to the sequentially stored files on disk. It is expected that *any secure chunk-level dedupe design should provide a read performance on par with plaintext CBD practice*.

Aiming to answer the above challenges, we design a chunk-based deduplication scheme for encrypted enterprise backup storage in the cloud. The core of the technique is the proposal of a *randomized oblivious key generation* (ROKG) protocol, which is simple by its design but powerful by its efficacy. Specifically, we are inspired by the observation that using randomized encryption will completely protect the low-entropy chunks albeit it, in turn, will outright incapacitate the deduplication. Similarly, we attempt to introduce the randomness into the chunk key generation. This gives us the desired asymmetry between security and performance, i.e. *resilient to multiple compromised clients*, compared to existing work, but only with *minimal dedupe performance loss*. We confine the proposed security and privacy preservation design to the enterprise internal network via setting up a key server in order to stay transparent to and compatible with the existing public cloud backup service. We further accelerate the key generation for frequent insensitive data by leveraging the *content-aware* deduplication. A *per-backup* rate-limiting strategy is also presented to further slow down the online brute-force attack without interfering the dedupe procedure. In addition, our design achieves *faster data restore speed* and *comparable space savings* for backup storage with plaintext CBD. We summarize the contributions as follows.

1) To the best of our knowledge, we are among the first to discuss the challenges of and solutions to securing chunk-based deduplication of encrypted backup storage as per the practical performance requirements.

2) We propose a randomized oblivious key generation algorithm, which can effectively reduce the risk of the information leakage by resilient to multiple compromised clients. We also enforce a per-backup rate limiting policy to slow down the online brute-force attack. Our presented scheme is provably secure in the malicious model.

3) We show that the efficiency of the online key generation for frequent insensitive data can be significantly improved by using content-aware deduplication technique. The experiment on the real-world dataset demonstrates a faster data restore speed while retaining an on-par deduplication effectiveness for backup storage with the plaintext CBD.

II. Background

A. Data Deduplication

Similar to data compression that identifies *intra-file* redundancy, deduplication is used to eliminate both *intra* and *inter* file duplicates. In general, chunk-based dedupe can capture "smaller" redundancy within files and thus often yields higher deduplication ratio $dr = \frac{originial\ dataset\ size}{stored\ dataset\ size}$, or the space reduction percentage $sr = 1 - 1/dr$ [12]. In storage backup scenario, the "original dataset" is an accumulated collection of all the data before deduplication from previous backup cycles. Further, if dedupe is allowed to be performed *cross users*, we usually can expect more space savings. On the other hand, dedupe occurring on the *server side* consumes more network bandwidth than the *client-side* dedupe, but with less privacy breach risk (see Sect. IV).

1) Chunking Algorithms: The data stream can be partitioned into *fixed-sized* chunks, which offers high processing rates and small computation overhead. However, it suffers from the boundary-shifting problem, where even a single bit added to the beginning of a file will result in different chunks [11]. A bit more CPU-intensive *variable-sized chunking* method can be used to address this problem. Briefly, this algorithm adopts a fixed-length sliding window to move onwards the data stream byte by byte. If the fingerprint (typically Rabin's fingerprint [14]), of the data segment covered by the window, satisfies a certain condition, this segment is marked as a partition point. The region between two consecutive partition points constitutes a chunk (see [10], [11] for detailed discussion). Variable-sized chunking provides users with more storage savings and is widely used in practice [11]. In addition, we can apply advanced *content-aware* chunking algorithms to identify duplicates on semantic information level [15], [16], [17], given the knowledge of file type, format, statistics information, etc. It turns out to be useful in speeding up the online chunk key generation (see Sect. V-A).

2) Chunk Fragmentation: A succinct chunk ID is computed by applying a hash function, such as SHA1[1], over this chunk. We can determine whether the chunk has already been stored by looking up a key-value index table that maintains unique chunk IDs and their corresponding chunk storage locations. To achieve high write performance, each unique chunk is not directly written into the storage; instead, it is stored into a fixed-sized container (typically 2MB or 4MB) in the cache and the whole container is flushed to the storage once it is full. To read a chunk from storage, the entire container storing the chunk is retrieved. Therefore, it is likely that data restoration needs to read the shared chunks physically dispersed over different containers. A higher chunk fragmentation means more severe physical dispersion, which ends up with read performance degradation [13], [18]. We can use the average number r of *containers read per MB* to measure the fragmentation level and evaluate the read performance by *speed factor* $1/r$ [18].

B. Convergent Encryption

CE is extensively used in secure dedupe systems [1], [2], [3], [4], [5], [6], [9] as a prominent instantiation of message-locked encryption (MLE) [7], [8]. More precisely, to encrypt a file f with CE, we first locally derive the CE key $k = h(f)$, where h is a secure hash function, e.g., SHA256. Next, we use any secure symmetric encryption Enc, such as AES128, with secret key k to obtain the ciphertext $c = Enc(k, f)$. Apparently, the deterministic encryption process will always generate the same ciphertext c for the same plaintext f and enable ciphertext deduplication. It is worth noting that CE

[1]Note that the security vulnerability of SHA1 is orthogonal to its application here in dedupe setting.

only provides security guarantees for unpredictable data and is inherently vulnerable to offline brute-force attack [4], [6]. In this work, we introduce a server-aided CE in the sense that the secret key is still derived from the target chunk but with the assistance of a dedicated key server (see Sect. V).

C. Blind RSA Signature

In a server-client model, blind signature allows the server to cryptographically sign the secure hash of a message from the client without disclosing the message content. In a blind RSA signature [19], let $\{N, e, d\}$ be a valid set of RSA parameters, where the modulus N is the product of two large primes p and q, $ed = 1 \bmod \varphi(N)$ and $\gcd(e, \varphi(N)) = 1$. $\varphi(N) = \text{lcm}(p-1, q-1)$. Then the public key is (e, N) and the private key is d. 1) $z \leftarrow \mathsf{MessageMask}(msg, r)$: The client prepares a random number $r \in \mathbb{Z}_n$ and a full domain hash $\text{H} : \{0,1\}^* \rightarrow \mathbb{Z}_n$. He masks his original message msg by $z = \text{H}(msg)r^e \bmod N$; 2) $\theta' \leftarrow \mathsf{Sign}(z)$: The server signs z with the private key and sends the signature $\theta' = z^d \bmod N$ back to the client; 3) $\theta \leftarrow \mathsf{Unmask}(\theta')$: On the client side, the intended signature on msg is derived from $\theta = \theta' r^{-1} \bmod N$ and can be verified by $\text{H}(msg) \stackrel{?}{=} \theta^e \bmod N$. In Section V, we will show how to build the ROKG protocol on top of the blind RSA signature and further improve its efficiency.

III. Related Work

In the literature, there are in general two approaches, i.e. server-aided and serverless schemes, to prevent the direct key derivation from the data by the client.

A. Server-aided Encryption Solutions

Server-aided solutions (including ours) is more suitable for the enterprise/organization network and transparent to the established deduplication services. Puzio *et al.* [2] proposed to use an honest proxy server to encrypt the CE-generated ciphertexts by the client before uploading them to a storage server. Their scheme claims to provide secure chunk-level deduplication but it is unclear how to mitigate online brute-force attack in the malicious model. By the adoption of an identity server, Stanek *et al.* in [3] presented a secure file-based deduplication scheme that prevents online brute-force attack from masqueraded clients. However, only non-private popular files can be deduplicated by using a threshold encryption. Bellare *et al.* [4] proposed a server-aided secure dedupe system in the enterprise setting. By blind RSA signature, the CE key can be obliviously generated. The offline brute-force attack is prevented since the compromised storage server cannot access the key server. However, the online attack is still possible by controlling a legitimate client. As a result, all the client's storage can be revealed by the attack. They applied a per-client file-based rate-limiting method to slow down the online attack.

B. Serverless Encryption Solutions

Duan [5] proposed to replace the role of a key server with clients using a modified Shoup RSA threshold signature scheme. It is unclear how to enforce any rate-limiting policy to slow down the brute-force attack. Xu *et al.* [9] proposed to deduplicate the message ciphertexts generated by randomized encryption. It only stores the first-uploaded file. For the same file uploading request, it provides the file encryption key to the user, which in turn is encrypted by the file. If the user indeed owns the file, he can derive the key and decrypt the file ciphertext. Brute-force attacks are avoided by assuming that the storage server is honest and cannot be compromised. In [6], the authors introduced a cross-user deduplication scheme. For an already stored file, a client executes a password-authenticated key exchange protocol with online clients who have previously uploaded the same file to obtain the CE key. Note that this process still needs to be coordinated by the storage server. Similar to [4], the offline brute-force attack is impossible because the CE key is not self-generated. They also adopt a per-file rate-limiting strategy to bound how many online protocol instances for each file can be invoked. Notice that these solutions cannot be directly integrated into the existing cloud storage services without substantial modification. Besides failing the protection of low-entropy chunks, applying the above-mentioned schemes to chunk-based deduplication will incur a considerable performance penalty in key generation and deduplication.

C. Other Security Aspects

The cross-user client-side deduplication may introduce side-channel attacks. By uploading a crafted file to the storage server and observing the deduplication process, the adversary can learn additional information about the file, e.g. whether it has been uploaded by other clients, etc. This is more devastating for predictable data. Harnik *et al.* [20] proposed a randomized threshold approach to alleviate such side channel attack. To avoid private data leakage by using a single hash, Halevi *et al.* [21] proposed a proof-of-ownership (PoW) framework to verify the ownership of the file that the client is trying to access. Recently, Li *et al.* [32] presented a practical attack to reveal the deduplicated ciphertext storage by frequency analysis due to the deterministic nature of CE/MLE. Along another research line, the authors in [22] proposed a CE key management scheme that applies deduplication over encryption keys and distributes the key shares across multiple key servers. Chen *et al.* in [31] proposed an MLE scheme also with the focus on key management in the CBD setting. However, they did not consider the protection of low-entropy chunk and practical dedupe performance.

IV. Problem Statement

A. System Model

There are three entities in our secure client-side cross-user deduplication system, key server ($\mathcal{KS}$), clients ($\mathcal{C}$'s) and public cloud storage server ($\mathcal{SS}$) as shown in Fig. 1. We consider a periodical file backup service provided to $\mathcal{C}$'s in an enterprise network. The key server $\mathcal{KS}$ is set up in charge of client authentication and chunk encryption key generation. Specifically, a client $\mathcal{C}_j$ performs the chunking algorithm on his backup data. $\mathcal{KS}$ authenticates $\mathcal{C}_j$ upon request and

generates the CE key k for each chunk ch of $\mathcal{C}_j$'s backup data in an oblivious manner. Then $\mathcal{C}_j$ encrypts the data chunks with the associated keys and uploads ciphertexts to $\mathcal{SS}$[2], such as Microsoft Azure Backup. $\mathcal{SS}$ stores the deduplicated incoming data stream in the corresponding containers before writing them into storage. As a result, the entire data protection phase, including key generation and data encryption, is transparent to $\mathcal{SS}$. $\mathcal{SS}$ only offers a basic and simple interface to its clients as in the plaintext data backup scenario.

B. Security Model

We focus on protecting the confidentiality of predictable data in this work because we can achieve semantic security for unpredictable data with CE. $\mathcal{KS}$ learns nothing about $\mathcal{C}_j$'s input chunk during the protocol execution. A compromised $\mathcal{SS}$ can launch *offline brute-force attack* by enumerating ciphertexts of predictable file candidates and comparing them with the target ciphertext in an offline manner. Although enterprise network is usually protected by enforcing rigorous security policies, we assume that it is possible for an external adversary to compromise a limited number of internal clients. Thus the adversary can perform an *online brute-force attack* by further accessing $\mathcal{KS}$.

We first define an ideal functionality $\mathcal{F}_{dedupe}$ of our scheme. The input of $\mathcal{F}_{dedupe}$:

- The client $\mathcal{C}_j$ has an input chunk ch;
- The key server $\mathcal{KS}$'s input is a chosen secret d_t;
- The cloud storage server $\mathcal{SS}$ has no input.

The output of $\mathcal{F}_{dedupe}$:

- $\mathcal{C}_j$ obtains the chunk key k;
- The output of $\mathcal{KS}$ is θ';
- $\mathcal{SS}$ gets the ciphertext $c = Enc(k, ch)$ and learns whether it has been stored.

We will prove our scheme secure in the malicious model if a probabilistic polynomial-time (PPT) adversary cannot distinguish the real-world execution of the proposed scheme and an ideal-world protocol that implements the functionality $\mathcal{F}_{dedupe}$ in the presence of a PPT simulator. In addition, we do not consider side-channel attacks, proof of ownership and key management in this study. Our system design will complement the current research [4], [6], [20], [21], [22], [31], [32]. Further, we assume that all the communication channels between $\mathcal{KS}$, $\mathcal{C}_j$ and $\mathcal{SS}$ are secure, and cannot be eavesdropped or tampered with by the adversary.

C. Design Goals

We devise a privacy-preserving chunk-based dedupe system aiming to achieve the following design goals. Pertaining to security, 1) realize the ideal functionality $\mathcal{F}_{dedupe}$ in the malicious model; 2) prevent offline brute-force attack by $\mathcal{SS}$; 3) mitigate online brute-force attack by slowing down its speed and providing multi-client compromise resilience. In the performance aspect, 4) realize efficient chunk encryption key generation; 5) our design should be comparable with the plaintext CBD with respect to performance, such as dedupe ratio and data restore speed.

[2]For simplicity, we omit the non-security steps, such as chunk ID generation and index table lookup on $\mathcal{SS}$.

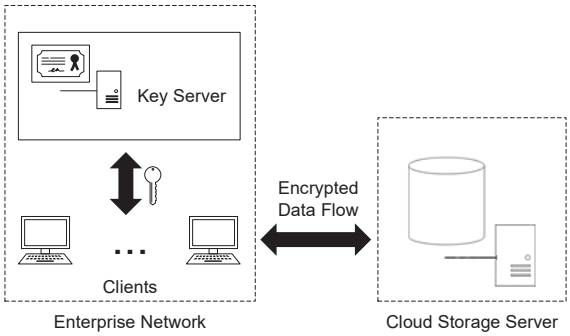


Fig. 1. Framework of the proposed scheme.

V. Protocol Design

In this section, we elaborate on our protocol design and provide discussion on the adopted techniques.

A. Randomized Oblivious Key Generation

Chunk encryption key can be generated by running a secure (oblivious) two-party computation between $\mathcal{C}_j$ and $\mathcal{KS}$, so that $\mathcal{KS}$ learns nothing on the $\mathcal{C}_j$'s input and algorithm output while $\mathcal{C}_j$ cannot infer $\mathcal{KS}$'s secret. In general, such desired protocol can be realized by any blind signature scheme. Here we use the widely-adopted blind RSA signature similar to prior work [4] and further introduce the randomness into the oblivious key generation.

1) Algorithm Definition: Let the hash functions $\mathrm{G} : \mathbb{Z}_n \rightarrow \{0,1\}^l$ and $\mathrm{H} : \{0,1\}^* \rightarrow \mathbb{Z}_n$. We define the ROKG algorithm as follows.

Definition 1: (ROKG algorithm) The proposed randomized oblivious key generation for a total of s clients in the system consists of four fundamental algorithms.

- $\mathsf{Setup}(\lambda_r, \lambda_n) \rightarrow (\{PK, MK\})$: The setup algorithm takes as input the security parameters λ_r and λ_n and outputs n sets of RSA parameters $\{(N_i, e_i, d_i) | 1 \leq i \leq n\}$. Thus, the public parameters are $PK = \{(N_i, e_i)\}$ and master secrets are $MK = \{d_i\}$.
- $\mathsf{ChObf}(ch, r, PK_i, \mathrm{H}) \rightarrow z$: This chunk obfuscation algorithm takes as input the chunk data ch, a random number r, the associated $PK_i = \{N_i, e_i\}$ and hash function H. It outputs the obfuscated chunk data z.
- $\mathsf{OKeyGen}(MK_i, z) \rightarrow \theta'$: This oblivious chunk key generation algorithm takes as input the associated master secret $MK_i = d_i$ for the client and obfuscated chunk z. It outputs the corresponding obfuscated chunk key θ'.
- $\mathsf{KeyRec}(\theta', \mathrm{H}, \mathrm{G}, PK_i, r) \rightarrow k$ or $\perp$: This chunk key recovery algorithm takes as input θ', hash functions G and H, the associated public parameter PK_i and the random number r. If θ' is successfully verified, it outputs the chunk encryption key k. Otherwise, it outputs $\perp$.

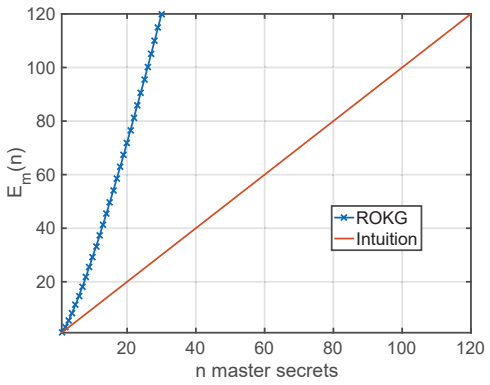


Fig. 2. $E_m(n)$ with the increased number n of the master secrets in the system.

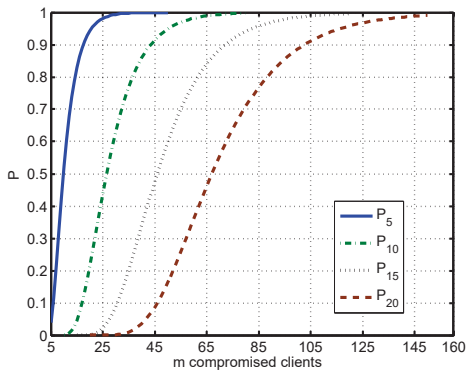


Fig. 3. $P_n(m)$ with m compromised clients when $n = 5,\ 10,\ 15,\ 20$.

2) ROKG Construction: In what follows, we provide concrete ROKG design.

System setup. At the setup phase, the key server $\mathcal{KS}$ calls the Setup algorithm to generate n pairs of $\{(PK_i, MK_i)\}$. PK is published to all the clients $\mathcal{C}$'s. MK is kept as the master secrets for the following protocol execution.

Client registration. Each new client $\mathcal{C}_j$ in the system needs to be authorized and registered by $\mathcal{KS}$ before he can request the chunk encryption key. Specifically, for the authorized $\mathcal{C}_j$, $\mathcal{KS}$ uniformly at random selects a master secret MK_i from MK and stores the tuple $(id(\mathcal{C}_j), i)$ on the user list. The selection i is then returned to $\mathcal{C}_j$.

Chunk data mask. For a chunk data ch, the client $\mathcal{C}_j$ calls the algorithm ChObf to obfuscate ch before sent to $\mathcal{KS}$. In particular, $\mathcal{C}_j$ chooses the corresponding $PK_i = \{N_i, e_i\}$ and a random number r. Then he masks the original chunk by $z = \mathrm{H}(ch)r^{e_i} \mod N_i$ and sends z to $\mathcal{KS}$.

Obfuscated chunk key generation. Upon receiving the key generation request from $\mathcal{C}_j$, the key server prepares the corresponding MK_i and PK_i by looking up the user list. $\mathcal{KS}$ then calls the OKeyGen algorithm to generate the obfuscated chunk key $\theta' = z^{d_i} \mod N_i$ and returns it to the client.

Key recovery. The client $\mathcal{C}_j$ invokes the algorithm KeyRec to derive the real chunk encryption key k. Specifically, he first unmasks θ' to $\theta = \theta' r^{-1} \mod N_i$. $\mathcal{C}_j$ then verifies θ by $\mathrm{H}(ch) \stackrel{?}{=} \theta^{e_i} \mod N_i$. If θ is valid, he can further recover the chunk encryption key $k = \mathrm{G}(\theta)$.

The proposed ROKG algorithm hides $\mathcal{C}_j$'s input chunk ch and actual output key k from $\mathcal{KS}$ while protecting $\mathcal{KS}$'s secrets MK from prying eyes of the client.

3) Asymmetry between Security Gain and Dedupability Loss: Intuitively, the security gain grows linearly with the increased number n of master secrets in the system but the dedupe effectiveness also degrades at the comparable rate. However, our proposed ROKG scheme brings us a counter-intuitive asymmetry property between the security gain and deduplication loss due to the characteristics of the accumulated storage backup. In other words, with the increased n, the growth rate of protection is much larger than that for dedupe performance loss. In what follows, we elaborate the impact of ROKG on these two aspects.

Multi-client compromise resilience. In prior work [2], [3], [4], once a legitimate client is compromised, the entire encrypted storage of all clients can be revealed using online brute-force attack by taking $\mathcal{KS}$ as a key oracle. In contrast, we introduce randomness into the key generation. Obviously, given any compromised $\mathcal{C}_j$ out of s clients in the system, the adversary can only infer at most $\frac{s}{n}$ clients' data on $\mathcal{SS}$ ($s \geq n$). On the other hand, the adversary is able to increase his advantage by compromising more clients. In previous works [4], [6], compromising one client suffices for the whole storage exposure. Here we care about the equivalent situation of revealing the storage under all n secrets by controlling m clients and quantify the leakage.

We define our security gain as $E_m(n)$, which is the expected number of clients to be compromised for accessing all n secrets. $E_m(n)$ can be denoted by $n(1 + \frac{1}{2} + \cdots + \frac{1}{n})$ from the insights of the coupon collector's problem [24]. By intuition, the growth rate of $E_m(n)$ is expected to be comparable with that of n. However, using ROKG gives us a much faster increase in $E_m(n)$ as shown in Fig. 2. Therefore, we can choose a relatively small n but stay resilient to more compromised clients. We also provide the accurate probability $P_n(m)$ for the case that the adversary compromises m clients in order to infer the storage under all n master secrets of $\mathcal{KS}$ ($m \geq n$). In general, there are n^m possible ways, in which we are interested in the number of functions from a set of m elements to a set of n elements. Such number can be denoted by $n!S(m,n)$. $S(m,n) = \frac{1}{n!}\sum_{j=0}^{n}(-1)^{n-j}\binom{n}{j}j^m$ is the Sterling number of the second kind [23]. Therefore, the probability is $P_n(m) = \frac{n!S(m,n)}{n^m}$. Given a fixed n, $P_n(m)$ grows as expected by compromising more clients shown in Fig. 3. It also exhibits that the whole dedupe system becomes more robust under the attack by increasing n.

In practice, we can tweak the parameter n to accommodate the real network scale. The number of compromised machines in an enterprise network usually depends on not only the company's size but also its security policies/controls. The typical infection percentage ranges from 0.1% to 18.5% [25]. To demonstrate the effectiveness of our protocol, we take into account a fairly protected small company of 100 employees with 10% or lower infection ratio (10 compromised clients). Thus, for $n = 5$ the adversary will succeed only with the

probability less than 50%. Note that we are free to adopt a larger n to further make $P_n(m)$ negligible.

Impact on deduplication. Indeed, the resulted threat isolation comes at the price of dedupe effectiveness loss. This is two folds. First, we study the case for one backup cycle. W.l.o.g, the stored data can be represented by $x + y$ under one secret. x is the size of data that cannot be deduplicated across all the users. y refers to the size of data that have been deduplicated. We consider the best case that all the users share the same data portion of y. Thus, $x + y$ is the lower bound of stored data size we can achieve in reality. By using n secrets in the system, the size of the stored data is $x + ny$. Compared to the dedupe ratio dr_1 under one key, the dedupe ratio using n keys is $dr_n = \frac{x+y}{x+ny} \cdot dr_1$. Obviously, the performance degradation does not follow the simple linearity, which is also demonstrated by our experiment (see Sect. VII). If x outsizes y significantly, selecting a small or moderate n will not introduce an obvious performance penalty. The example may be that the backup contains data types not naturally suited for deduplication, such as compressed files commonly seen in the archive storage, the rich media data (e.g. videos, images). Therefore, our scheme can provide better security guarantee in this situation. Otherwise, non-negligible dedupe loss is expected for this one-time backup scenario. On the other hand, the periodic backup service will eventually give rise to a high dedupe performance with our scheme. This is because the size of the accumulated backup storage is a more dominant factor in the dedupe ratio computation compared to the orders of magnitude smaller n. Thus, we can take advantage of this asymmetry to achieve stronger privacy protection with a larger n while enjoying comparable space savings with the plaintext CBD. This analysis is consistent with our experiment (see Sect. VII). In addition, our scheme enables better read performance (see Sect. V-C).

4) Efficiency Improvement for Frequent Insensitive Data: We observe that files sharing the similar contents or with the same data type, e.g., .pdf, .doc, may contain identical data fields. Intuitively, if we extract these immutable parts and utilize them as a file fingerprint, we can accelerate the key generation significantly. In this case, we modify the original ROKG protocol for the frequent insensitive data as follows.

The setup and client registration remain the same. $\mathcal{C}_j$ first adopts the content-aware deduplication [15], [16], [17] to identify the common data parts for his files with the same format or data type. For instance, in Fig. 4, given the extracted common data chunks F_1, F_2, and F_3, we can compute the file format fingerprint $h_f = \mathrm{H}(F_1||F_2||F_3)$. Instead of running the remaining algorithms, i.e. ChObf, OkeyGen, and KeyRec for each chunk, $\mathcal{C}_j$ uses the fingerprint h_f as the input to get the file format key k_f. Subsequently, $\mathcal{C}_j$ produces the chunk key $k = \mathrm{G}(k_f||ch)$ offline for all the chunks ch in the file with the same fingerprint. Therefore, we have a constant computation and communication overhead for the modified ROKG protocol. For files containing sensitive information, $\mathcal{C}_j$ still needs to run the online protocol per chunk with $\mathcal{KS}$ to stay more resilient to the brute-force attack.

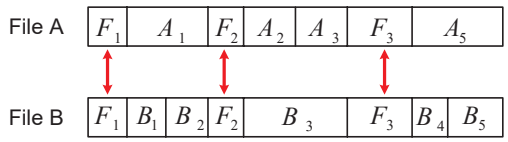


Fig. 4. F_1, F_2, and F_3 are immutable parts identified by content-aware chunking algorithm in two file copies A and B of the same file format.

B. Slowing down Online Brute-force Attack

Online brute-force attack can be launched by compromising a legitimate client and interacting with the key server to obtain the chunk encryption key. Completely preventing such attack is still an open problem. Using our ROKG design only partially mitigate this issue. On the other hand, rate-limiting strategy is broadly used to *slow down* this online attack in file-based dedupe scenario [4], [6]. We propose to enforce a *per-backup* rate-limiting policy in the chunk-based dedupe system, which is inspired by the observed features of storage backup in practice. Specifically, given the projected backup data size and expected chunk size, we set a budget $q = \frac{projected\ backup\ data\ size}{expected\ chunk\ size}$ for each client to bound the number of requests that are allowed to be processed by $\mathcal{KS}$ during the prescribed time window, e.g. 2:00 – 3:00 AM every Tuesday. Otherwise, $\mathcal{KS}$ will not respond to the client.

This policy is made based on the following observations. First, the enterprise backup workloads usually exhibit periodicity, i.e., they follow the scheduled time window and update cycle. Moreover, it is expected that the content and size of the periodical backup data from an enterprise user does not change rapidly [10]. For example, a weekly 2GB OS snapshot of a client's machine is backed up to the cloud storage with the expected chunk size 8KB. We can estimate a weekly backup budget $q = 250,000$ for each client. We may also set an additional buffer to tolerate the error and ensure the success of the backup. We assume that any attempt to use the budget for the attack without actually storing the data, or only storing a portion below the budget will be detected in a post auditing process. In addition, our approach is supposed to work with both full and incremental backup (only storing deltas between files) scenarios. Note that the rate-limiting strategy may not be fully compatible with the proposed content-aware key generation mechanism because the adversary can circumvent the online restriction by offline computation. Thus, it is desired to enforce the policy for sensitive data.

C. Improving Data Restore Speed

There are several reasons why we are concerned about read performance even in the backup storage. First of all, data restore speed is considered critical for crash/corruption recovery, where higher read speed results in shorter recovery window time. Furthermore, we need to reconstruct the original data stream (more frequent than user-triggered data retrieval) for staging the backup data streams to archive storage in light of limited capacity of deduplication storage [26].

Despite the reduced dedupe ratio, the proposed scheme will naturally enable better read performance for a user as we allow a duplicate chunk copy under one secret to be kept in the storage without referring it to an existing copy under another secret in an old container. As a result, we trade off deduplication for faster user backup restore speed, which happens to reflect a similar optimization philosophy in plaintext dedupe research [15], [26], [27]. We can further improve read performance by adopting a reconstruction-aware chunk placement mechanism to enforce a high spatial locality for chunks. Specifically, the system maintains a set of dedicated chunk containers $cnt_{i,j}$ in the cache for each $\mathcal{KS}$ secret d_i, where $1 \leq i \leq n$ and j indicates a distinct container for the same key[3]. We achieve high spatial locality by storing $cnt_{i,j}$ in separate locations of the disk according to i, such as in different partitions. Therefore, chunks under the same $\mathcal{KS}$ secret are stored close to each other. When restoring a client's data, read access is only restricted to a limited scope of the disk instead of random accessing the whole storage.

We argue that the proposed chunk placement will not disclose the additional information except what has been learned by the adversary. In particular, while improving the read performance, the adversary may identify clients under the same $\mathcal{KS}$ secret by observing their chunks stored in the same set of containers $cnt_{t,j}$. However, such information leakage is inevitable in any dedupe system, which the adversary on $\mathcal{SS}$ always knows from deduplication process. Furthermore, we can leverage the encrypted data search techniques to realize the secure chunk retrieval and verification [28] or combine ORAM to hide the access pattern [29], which are interesting future research directions.

VI. Security Analysis

In this section, we show that the presented scheme accomplishes our security goals. As we discussed, our proposal alleviates the online brute-force attack even when multiple clients are compromised and we can diminish the attack efficiency by the *per-backup* rate-limiting strategy. In what follows, we show that $\mathcal{F}_{dedupe}$ and offline brute-force attack prevention are achievable as well.

Theorem 1: Our secure chunk-based deduplication protocol computing the ideal functionality $\mathcal{F}_{dedupe}$ is secure in the malicious model if the blind RSA signature is secure in the random oracle model and the hash function G is modeled as a random oracle.

Proof: (Sketch) Assume the blind RSA signature protocol to be an oracle that takes in participants' inputs and then sends outputs to them. For simplicity, we only consider one $\mathcal{KS}$ secret in the system, whose security can be easily extended to the multi-secret situation. Suppose that there is a simulator $\mathcal{S}$ for the corrupted parties in the ideal world. $\mathcal{S}$ can access $\mathcal{F}_{dedupe}$ in the ideal world and record message transcript from the protocol execution in the real world. Therefore, the adversary $\mathcal{A}$ cannot distinguish the view $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda_r)$ constructed by $\mathcal{S}$ in the ideal world from the view $\mathbf{Real}_{\mathcal{A}}(\lambda_r)$ in the real-world protocol execution.

Corrupted $\mathcal{C}_j$: Assume that $\mathcal{SS}$ and $\mathcal{KS}$ are honest in this case. Simulator $\mathcal{S}$ records the calls $\mathcal{C}_j$ makes to the blind RSA signature with the input chunk ch. It invokes the ideal functionality $\mathcal{F}_{dedupe}$ with the same ch. $\mathcal{S}$ also records the output θ from the blind RSA signature and keeps a list $\{(\theta, k)\}$. If $\mathcal{S}$ receives a θ that appears on the list, it returns the corresponding k as the chunk key. Otherwise, k is a random number and $\mathcal{S}$ writes it back onto the list. In the end, $\mathcal{F}_{dedupe}$ also outputs a chunk key k_Δ for ch and θ'_Δ. $\mathcal{S}$ may simulate the message transcript as follows. It sets $r = \frac{\theta'_\Delta}{\theta}$. z_Δ is simulated as $r^e\theta^e$ and $h_\Delta = \theta^e$. If $\mathcal{C}_j$ behaves honestly, k is equal to k_Δ. On the other hand, $\mathcal{A}$ can deviate from the protocol by modifying his inputs and replacing elements that are sent to $\mathcal{S}$. In this case, θ and k are random numbers in light of the RSA signature and random oracle G. $\mathcal{S}$ can simulate the message transcript similar to the above. The message transcripts cannot be computationally distinguished by adversary $\mathcal{A}$ in the real and ideal worlds. Thus, $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda_r)$ and $\mathbf{Real}_{\mathcal{A}}(\lambda_r)$ are identically distributed.

Corrupted $\mathcal{KS}$: Assume that $\mathcal{C}_j$ and $\mathcal{SS}$ are honest. $\mathcal{F}_{dedupe}$ outputs θ'_Δ for $\mathcal{KS}$. $\mathcal{S}$ can simulate the incoming message z_Δ similarly to the above. $\mathcal{KS}$ can deviate from the designated protocol execution by replacing the signature, which, however, will only pass the client-side verification with negligible probability given the unforgeability of blind RSA signature. Thus, $\mathcal{A}$ still cannot distinguish $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda_r)$ and $\mathbf{Real}_{\mathcal{A}}(\lambda_r)$.

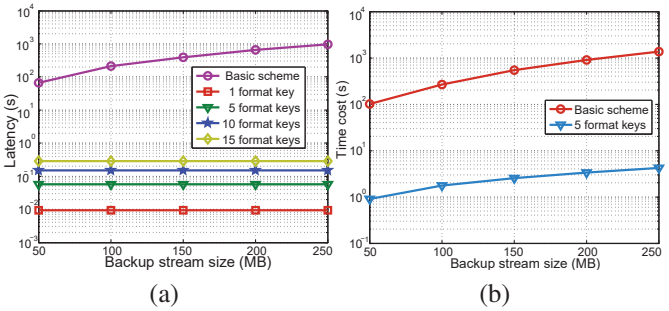
Corrupted $\mathcal{SS}$: Assume that both $\mathcal{C}_j$ and $\mathcal{KS}$ are honest. By providing $\mathcal{F}_{dedupe}$ with the input ch and d_i, $\mathcal{S}$ receives the chunk ciphertext $c_\Delta = Enc(k_\Delta, ch)$. If ch exists, the ideal-world c_Δ is the same as c in the real world. Otherwise, the chunk key and ciphertext are random in both the real and ideal worlds. $\mathcal{SS}$ can deviate from the protocol by modifying the ciphertext, which may result in a failed chunk deduplication. Thus, $\mathcal{A}$ cannot distinguish $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda_r)$ and $\mathbf{Real}_{\mathcal{A}}(\lambda_r)$. ■

According to Theorem 1, the proposed protocol securely computes the ideal functionality $\mathcal{F}_{dedupe}$. In addition, the adversary (in the case of corrupted $\mathcal{SS}$) cannot generate the encryption key from chunk data itself or access $\mathcal{KS}$, thereby preventing the offline brute-force attack.

VII. Performance Evaluation

We implement our secure chunk-based deduplication system on the real-world backup storage from File systems and Storage Lab at Stony Brook University [30]. We focus on the 2013 MacOS Snapshots dataset collected on a Mac OS X Snow Leopard server with 54 users. There are 249 snapshots (daily backup) in total with the duration of 11 months, and each snapshot is generated by using variable-sized chunking with an average chunk size of 8KB. To simulate our enterprise backup setting, we synthesize each individual user's daily backup by extracting his files from the snapshot and incorporate data of `UID-0` as the base file system. The total size of the backup storage we considered here before deduplication is roughly 463 TB. We also set the size of the chunk container and LRU

[3]We need an additional 1-byte field *cnt_num* in the chunk metadata to mark which set of containers this chunk should go to.



(a) (b)

Fig. 5. (a) Latency and (b) Client-side computation overhead for the online key generation protocol with the increased size of backup stream of a client. We invoke an online protocol instance for each chunk in our basic scheme.

cache as 4 MB and 512 MB respectively. We do not use the relevant optimization technique, such as parallelization. The corresponding experimental results are an average of 100 trials.

The existing secure chunk-based designs [2], [22], [31], with the different research focus, do not consider the protection of low-entropy data in the presence of a strong attacker and the practical deduplication performance, such as fragmentation level. Their performance should be roughly the same as that of plaintext CBD in terms of dedupe ratio and data restore speed because they heuristically keep unique chunk copy in the system. We will not explicitly mention them hereafter and only compare the proposed scheme with plaintext CBD instead (see Sect. VII-B and VII-C).

A. Online Key Generation

We use Python to implement our TCP-based randomized oblivious key generation protocol between the key server $\mathcal{KS}$ and client $\mathcal{C}_j$. The server machine is equipped with a 3.1 GHz AMD FX 8120 processor and 32GB DDR3 memory. On the same LAN, the client machine has an Intel i3-2120 processor with 12GB memory. The blind RSA signature is implemented with RSA1024 and SHA256. CE is instantiated by CTR[AES128].

1) Latency: We measure the latency incurred by the proposed online protocol, which is defined as the time between that $\mathcal{C}_j$ sends out the request to and receives the response from $\mathcal{KS}$. As shown in Fig. 5(a), the overhead of the basic scheme without optimizing the chunk key generation is linear in the size of the client's data. A large size backup data stream produces more chunks, which will invoke more per-chunk key generation protocol instances. Fig. 5(a) also shows that the protocol latency is dramatically reduced and becomes constant if we resort to the content-aware chunking and only run the protocol to generate the format key k_f. Thus, the latency merely depends on the number of distinct file formats in the backup stream of the client (supposing all are predictable files), regardless of the size.

2) Client-side Cost: We also measure the additional client-side cost due to the ROKG protocol and convergent encryption. As shown in Fig. 5(b), the overhead is mainly contingent on the size of the backup data stream. It is worth noting that this cost still demonstrates the linearity with the total backup data size even using the proposed efficient key generation algorithm. This is because client still needs to carry out the offline hashing for each chunk key, and encrypts them individually. Again, it shows a significant performance advantage by the efficient key generation protocol.

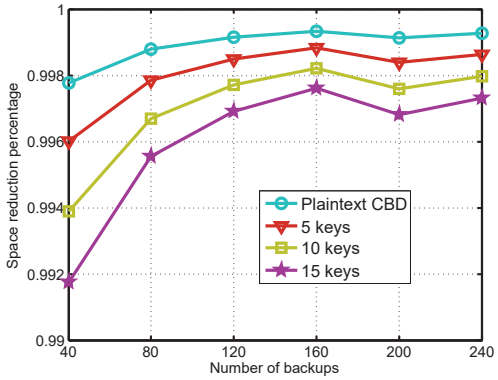
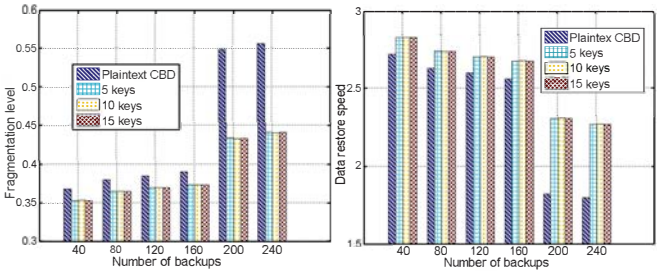


Fig. 6. Space savings sr with the increased number of backup storage in the cloud. Each backup includes snapshots of 54 users' machines. The comparison is drawn between the plaintext CBD, and our scheme using 5, 10, and 15 $\mathcal{KS}$ secret keys.



(a) (b)

Fig. 7. (a) Fragmentation level r and (b) data restore speed $1/r$ with the increased number of backups in the cloud. Both fragmentation level and read performance are measured by recovering the backup dataset of a randomly selected user. All the comparisons are drawn between the plaintext CBD, and our scheme using 5, 10, and 15 $\mathcal{KS}$ secret keys.

B. Deduplication Effectiveness

We develop a deduplication simulator with C code to measure the corresponding dedupe performance of the proposed scheme and compare our protocol with plaintext CBD.

Adhering to our previous analysis (see Sect. V-A3), for a single backup attempt, for example, the $40th$ backup in Fig. 6, the dedupe ratio dr_5 with 5 $\mathcal{KS}$ secrets is about half of dr_1 in the plaintext CBD, which exhibits a non-linear performance loss with n. Fig. 6 also shows that the space reduction percentage sr is sensitive to the number of backups and increases as periodic backup service continues. With more backup date added, sr gradually approaches the plaintext dedupe performance regardless of the number of master secrets used in the system because the size of the accumulated "original dataset" dominates the computation for sr. As a result, we can adopt more $\mathcal{KS}$ secrets to achieve stronger privacy protection with a minimal dedupability loss.

C. Fragmentation

We focus on the fragmentation/read performance comparison with plaintext CBD because of the sequentially writ-

ten/read files and defragmentation schedule in file-based deduplication. Although our design shows a slight loss of dedupability so as to achieve the desired security objectives, the chunk fragmentation level r for user backup is also reduced, thereby the increased data restore speed shown in Fig. 7(a) and Fig. 7(b) respectively.

We neglect the actual low-level data placement on disk (i.e. our high spatial locality design in Sect. V-C), and other common optimization techniques (e.g. large container/cache). We use the same simulator to study the impact of fragmentation on a user's backup data caused by our security design only. The result, as shown in Fig. 7(a), again validates the importance of the fragmentation issue in the chunk-based deduplication that the more data added to the storage, the more severe the chunk fragmentation level. Fig. 7(a) also shows that our design can maintain a lower fragmentation level than plaintext CBD. In addition, the number of $\mathcal{KS}$ secret keys used in the system has a negligible impact on the chunk fragmentation. Note that we can enjoy substantial fragmentation reduction with the increased size of backup storage. Fig. 7(b) accordingly shows that the proposed scheme achieves better user backup read performance than plaintext CBD. As a result, we can use more $\mathcal{KS}$ secret keys to realize faster restore speed and stronger privacy guarantees at the same time.

VIII. Conclusion

We in this work discuss and address challenges in designing a data deduplication system at the chunk level. The impact of online brute-force attack can be substantially alleviated by allowing clients to invoke the proposed randomized oblivious key generation protocol with a key server and enforcing a per-backup rate-limiting policy. We also exploit the content-aware deduplication technique to further improve the efficiency of the online key generation. Our scheme is on par with the plaintext practice in terms of the deduplication performance while gaining better security guarantees compared to the existing work.

Acknowledgments

This work was supported in part by US National Science Foundation grants CNS-1446478 and CNS-1443889.

References

[1] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system", in *Proc. of IEEE ICDCS*, pp. 617–624, 2002.

[2] P. Puzio, R. Molva, M. Onen, and S. Loureiro, "ClouDedup: Secure deduplication with encrypted data for cloud storage", in *Proc. of IEEE CloudCom*, pp. 363–370, 2013.

[3] J. Stanek, A. Sorniotti, E. Androulaki, and L. Kencl, "A secure data deduplication scheme for cloud storage", in *Financial Cryptography and Data Security*, pp. 99–118, 2014.

[4] M. Bellare, S. Keelveedhi, and T. Ristenpart, "DupLESS: Server-aided encryption for deduplicated storage", in *Proc. of USENIX Security*, pp. 179–194, 2013.

[5] Y. Duan, "Distributed key generation for encrypted deduplication: Achieving the strongest privacy", in *Proc. of ACM CCSW*, pp 57–68, 2014.

[6] J. Liu, N. Asokan, and B. Pinkas, "Secure deduplication of encrypted data without additional independent servers", in *Proc. of ACM CCS*, pp. 874–885, 2015.

[7] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication", *Advances in Cryptology-EUROCRYPT*, pp. 296–312, 2013.

[8] M. Abadi, D. Boneh, I. Mironov, A. Raghunathan, and G. Segev, "Message-locked encryption for lock-dependent messages", *Advances in Cryptology-CRYPTO*, pp. 374-391, 2013.

[9] J. Xu, E.-C. Chang, and J. Zhou, "Weak leakage-resilient client-side deduplication of encrypted data in cloud storage", in *Proc. of ACM ASIACCS*, pp. 195–206, 2013.

[10] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication", *ACM TOS*, vol. 7, no. 4, p. 14, 2012.

[11] J. Paulo and J. Pereira, "A survey and classification of storage deduplication systems", *ACM CSUR*, vol. 47, no. 1, p. 11, 2014.

[12] M. Dutch, "Understanding data deduplication ratios", *SNIA Data Management Forum*, http://www.snia.org/sites/default/files/Understanding_Data_Deduplication_Ratios-20080718.pdf, 2008.

[13] M. Kaczmarczyk, M. Barczynski, W. Kilian and C. Dubnicki, "Reducing impact of data fragmentation caused by in-line deduplication", in *Proc. of ACM SYSTOR*, p. 15, 2012.

[14] A. Z. Broder, "Some applications of Rabins fingerprinting method", *Sequences II: Methods in Communications, Security and Computer Science*, pp. 143–152, 1993.

[15] D. R. Bobbarjung, S. Jagannathan, and C. Dubnicki, "Improving duplicate elimination in storage systems", *ACM TOS*, vol. 2, no. 4, pp. 424–448, 2006.

[16] C. Liu, Y. Lu, C. Shi, G. Lu, D. H. Du, and D. S. Wang, "ADMAD: Application-driven metadata aware de-duplication archival storage system", in *Proc. of IEEE SNAPI*, pp. 29–35, 2008.

[17] G. Lu, Y. Jin, and D. H. Du, "Frequency based chunking for data deduplication", in *Proc. of IEEE MASCOTS*, pp. 287–296, 2010.

[18] M. Lillibridge, K. Eshghi, and D. Bhagwat, "Improving restore speed for backup systems that use inline chunk-based deduplication", in *Proc. of Usenix Fast*, pp. 183–197, 2013.

[19] D. Chaum, "Blind signatures for untraceable payments", *Advances in Cryptology-CRYPTO*, pp. 199–203, 1983.

[20] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage", in *Proc. of IEEE S&P*, vol. 8, no. 6, pp. 40–47, 2010.

[21] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems", in *Proc. of ACM CCS*, pp. 491–500, 2011.

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

[23] R. P. Stanley, "What is Enumerative Combinatorics?", *Enumerative Combinatorics*, pp. 1–63, 1986.

[24] B. Dawkins, "Siobhan's problem: the coupon collector revisited", *The American Statistician*, vol. 45, no. 1, pp. 76–82, 1991.

[25] "State of Infection Report – Q2 2014", *Damballa*, http://landing.damballa.com/state-infections-report-q2-2014.html, 2014.

[26] Y. Nam, D. Park, and D. H. Du, "Assuring demanded read performance of data deduplication storage with backup datasets", in *Proc. of IEEE MASCOTS*, pp. 201–208, 2012.

[27] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti, "iDedup: Latency-aware, inline data deduplication for primary storage", in *Proc. of USENIX FAST*, pp. 1–14, 2012.

[28] W. Sun, X. Liu, W. Lou, Y. T. Hou, and H. Li, "Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data", in *Proc. of IEEE INFOCOM*, pp. 2110–2118, 2015.

[29] O. Goldreich, "Towards a theory of software protection and simulation by oblivious rams", in *Proc. of TC*, pp. 182–194, 1987.

[30] "Traces and snapshots public archive", *FSL*, http://tracer.filesystems.org, 2012.

[31] R. Chen, Y. Mu, G. Yang, and F. Guo, "BL-MLE: Block-Level Message-Locked Encryption for Secure Large File Deduplication", *IEEE TIFS*, vol. 10, no. 12, pp. 2643–2652, 2015.

[32] J. Li, C. Qin, P. Lee, and X. Zhang "Information Leakage in Encrypted Deduplication via Frequency Analysis", in *Proc. of IEEE/IFIP DSN*, pp. 2110–2118, 2017.