Information Leakage in Encrypted Deduplication via Frequency Analysis

Jingwei Li[1], Chuan Qin[2], Patrick P. C. Lee[2], Xiaosong Zhang[1]
[1]Center for Cyber Security, University of Electronic Science and Technology of China
[2]Department of Computer Science and Engineering, The Chinese University of Hong Kong
lijw1987@gmail.com, chintran@live.cn, pclee@cse.cuhk.edu.hk, johnsonzxs@uestc.edu.cn

***Abstract*—Encrypted deduplication seamlessly combines encryption and deduplication to simultaneously achieve both data security and storage efficiency. State-of-the-art encrypted deduplication systems mostly adopt a deterministic encryption approach that encrypts each plaintext chunk with a key derived from the content of the chunk itself, so that identical plaintext chunks are always encrypted into identical ciphertext chunks for deduplication. However, such deterministic encryption inherently reveals the underlying frequency distribution of the original plaintext chunks. This allows an adversary to launch frequency analysis against the resulting ciphertext chunks, and ultimately infer the content of the original plaintext chunks.**

In this paper, we study how frequency analysis practically affects information leakage in encrypted deduplication storage, from both attack and defense perspectives. We first propose a new inference attack that exploits chunk locality to increase the coverage of inferred chunks. We conduct trace-driven evaluation on both real-world and synthetic datasets, and show that the new inference attack can infer a significant fraction of plaintext chunks under backup workloads. To protect against frequency analysis, we borrow the idea of existing performance-driven deduplication approaches and consider an encryption scheme called MinHash encryption, which disturbs the frequency rank of ciphertext chunks by encrypting some identical plaintext chunks into multiple distinct ciphertext chunks. Our trace-driven evaluation shows that MinHash encryption effectively mitigates the inference attack, while maintaining high storage efficiency.

I. Introduction

To manage massive amounts of data in the wild, modern storage systems employ *deduplication* to eliminate content duplicates and save storage space. The main idea of deduplication is to store only data copies, called *chunks*, that have unique content among all already stored chunks. Field studies have demonstrated that deduplication achieves significant storage savings in production environments, for example, by 50% in primary storage [33] and up to 98% in backup storage [45]. Deduplication is also adopted by commercial cloud storage services (e.g., Dropbox, Google Drive, Bitcasa, etc.) for cost-efficient outsourced data management [22], [34].

In the security context, combining encryption and deduplication, referred to as *encrypted deduplication*, is essential for protecting against content leakage in deduplication storage. Conventional (symmetric) encryption is incompatible with deduplication, as it requires that users encrypt data with their own distinct secret keys, thereby encrypting duplicate plaintext chunks into distinct ciphertext chunks. To preserve deduplication effectiveness, encrypted deduplication ensures

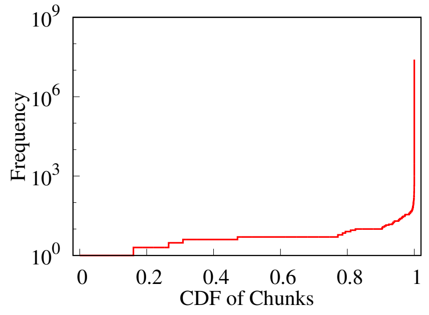


Fig. 1. Frequency distribution of chunks in the FSL dataset.

that ciphertext chunks originated from duplicate plaintext chunks can still be deduplicated. *Message-locked encryption (MLE)* [9] formalizes a cryptographic primitive to address the issue, by encrypting each chunk with a secret key that is derived from the chunk itself via some one-way function. For example, convergent encryption [16] is one classical instantiation of MLE by deriving the secret key through the hash of a chunk. On top of MLE, several storage systems address additional security issues, such as brute-force attacks [8], key management failures [17], side-channel attacks [28], and access control [38].

However, we argue that existing MLE implementations (see Section VIII) still cannot fully protect against content leakage, mainly because their encryption approaches are *deterministic*. That is, each ciphertext chunk is encrypted by a key that is deterministically derived from the original plaintext chunk. Thus, an adversary, which can be malicious users or storage system administrators, can analyze the frequency distribution of ciphertext chunks and infer the original plaintext chunks based on classical frequency analysis [36]. In addition, practical storage workloads often exhibit non-uniform frequency distributions in terms of the occurrences of chunks with the same content, thereby allowing the adversary to accurately differentiate chunks by their frequencies in frequency analysis. Figure 1 depicts the frequency distribution of chunks in the real-world FSL dataset used in our evaluation (see Section V). We observe that the frequency distribution varies significantly: 99.8% of chunks occur less than 100 times, while around 30 chunks occur over 10,000 times.

The deterministic nature of MLE makes encrypted deduplication vulnerable to frequency analysis. In the simplest form of the attack, an adversary first obtains prior knowledge of frequency distributions of plaintext chunks (e.g., by unintended data release [6] or data breaches [20]), counts the frequencies of all ciphertext chunks, and finally infers their corresponding plaintext chunks based on the frequency distribution of ciphertext chunks. While previous studies [3], [7] have addressed the possibility of launching frequency analysis against MLE-based storage and also proposed cryptographic mechanisms to mitigate the issue, their investigations are theoretically driven. The *practical* implications of frequency analysis against encrypted deduplication remain unexplored.

Contributions: In this paper, we conduct an in-depth study of how frequency analysis practically affects information leakage in encrypted deduplication. Our study spans both attack and defense perspectives, and is specifically driven by the characteristics of storage workloads in deduplication systems.

On the attack side, we propose a *locality-based* attack to enhance the severity of classical frequency analysis by exploiting *chunk locality*, which is prevalent in backup workloads. Chunk locality states that chunks are likely to re-occur together with their neighboring chunks across backups. In practice, changes to backups often appear in few clustered regions of chunks, while the remaining regions of chunks will appear in the same order in previous backups. Previous studies have exploited chunk locality to improve deduplication performance [30], [47], [49]. Here, we adapt this idea from a security perspective into frequency analysis: if a plaintext chunk M corresponds to a ciphertext chunk C, then the neighboring plaintext chunks of M are likely to correspond to the neighboring ciphertext chunks of C. Our trace-driven evaluation, using both real-world and synthetic datasets, shows that the locality-based attack can identify significantly more ciphertext-plaintext pairs than classical frequency analysis. For example, for the real-world FSL dataset, we find that the locality-based attack can infer a fraction of 17.8% of the latest backup data, while the basic attack based on the direct application of classical frequency analysis can only infer 0.0001% of data. In addition, if a limited fraction (e.g., 0.2%) of plaintext information of the latest backup is leaked, the inference rate of the locality-based attack can reach up to 27.1%.

On the defense side, our key insight of combating frequency analysis is to disturb the frequency ranking of ciphertext chunks. To this end, we borrow the idea from previous performance-driven deduplication approaches [10], [38], [47]. We consider an encryption scheme called *MinHash encryption*, which derives an encryption key based on the minimum fingerprint over a set of adjacent chunks, such that some identical plaintext chunks can be encrypted into multiple distinct ciphertext chunks. Our trace-driven evaluation shows that MinHash encryption effectively mitigates the locality-based attack, while maintaining high storage efficiency as demonstrated in previous deduplication approaches. For example, for the real-world dataset, if we repeat our attack evaluation that 0.2% of plaintext information of the latest backup is leaked, MinHash encryption can now suppress the inference rate of the locality-based attack to below 0.45%; meanwhile, it achieves a storage saving of up to 83.61%, which is only 3-4% less than that of original chunk-based deduplication.

II. Basics

Following Section I, we elaborate the basics of deduplication, encrypted deduplication, and frequency analysis.

A. Deduplication

Deduplication can be viewed as a coarse-grained compression technique to save storage space. While it can operate at the granularities of files or chunks, this paper focuses on chunk-based deduplication as it achieves more fine-grained redundancy elimination. Specifically, a storage system partitions input data into variable-size chunks through content-defined chunking (e.g., Rabin fingerprinting [39]), which identifies chunk boundaries that match specific content patterns so as to remain robust against content shifts [18]. We can configure the minimum, average, and maximum chunk sizes in content-defined chunking for different granularities. After chunking, each chunk is identified by a *fingerprint*, which is computed from the cryptographic hash of the content of the chunk. Any two chunks are said to be identical if they have the same fingerprint, and the collision probability that two non-identical chunks have the same fingerprint is practically negligible [11]. Deduplication requires that only one physical copy of identical chunks is kept in the storage system, while any identical chunk refers to the physical chunk via a small-size reference.

To check if any identical chunk exists, the storage system maintains a *fingerprint index*, a key-value store that holds the mappings of all fingerprints to the addresses of physical chunks that are currently stored. For each file, the storage system also stores a *file recipe* that lists the references to all chunks of the file for future reconstruction.

B. Encrypted Deduplication

Encrypted deduplication ensures that all physical chunks are encrypted for confidentiality (i.e., data remains secret from unauthorized users and even storage system administrators), while the ciphertext chunks that are originated from identical plaintext chunks can still be deduplicated for storage savings. As stated in Section I, message-locked encryption (MLE) [9] is a formal cryptographic primitive to achieve encrypted deduplication, in which each chunk is encrypted by a symmetric key that is derived from the chunk itself. Thus, identical plaintext chunks will be encrypted into the identical ciphertext chunks, thereby preserving deduplication effectiveness.

MLE is inherently vulnerable to the offline brute-force attack [9], which allows an adversary to determine which plaintext chunk is encrypted into an input ciphertext chunk. Suppose that the adversary knows the set of chunks from which the plaintext chunk is drawn. Then it can launch the brute-force attack as follows: for each chunk from the set, it finds the chunk-derived key (whose key derivation algorithm is supposed to be publicly available), encrypts the chunk with the chunk-derived key, and finally checks if the output ciphertext chunk is identical to the input ciphertext chunk. If so, the plaintext chunk is the answer. Thus, MLE can only achieve security for *unpredictable* chunks [9], meaning that the size of the set of chunks is sufficiently large, such that the brute-force attack becomes infeasible.

To protect against the brute-force attack, DupLESS [8] realizes *server-aided MLE*, which outsources MLE key management to a dedicated *key manager* that is only accessible by authenticated clients. Each authenticated client first queries the key manager for the chunk-derived key. Then the key manager computes and returns the key via a deterministic key derivation algorithm that takes the inputs of both the chunk fingerprint and a system-wide secret maintained by the key manager itself. This makes the resulting ciphertext chunks appear to be encrypted by a random key from the adversary's point of view. In addition, the key manager limits the rate of key generation to slow down any online brute-force attack for querying the encryption key. If the key manager is secure from adversaries, server-aided MLE ensures security even for predictable chunks; otherwise, it still maintains security for unpredictable chunks as in original MLE [9].

Most existing implementations of MLE-based encrypted deduplication, either realized as convergent encryption or server-aided MLE, follow the notion of deterministic encryption, which ensures that identical plaintext chunks always form identical ciphertext chunks to make deduplication possible. Thus, they inherently become vulnerable to frequency analysis as we show in this paper. Some encrypted deduplication designs are based on non-deterministic encryption [3], [7], [9], [31], yet they still keep deterministic components [9], incur high performance overhead [31], or require cryptographic primitives that are not readily implemented [3], [7]. We elaborate the details in Section VIII.

C. Frequency Analysis

Frequency analysis [32] is a classical inference attack that has been historically used to recover plaintexts from substitution-based ciphertexts, and is known to be useful for breaking deterministic encryption. In frequency analysis, an adversary has access to a set of plaintexts and a set of ciphertexts, and its attack goal is to relate each ciphertext to the plaintext in both sets. To launch the attack, the adversary ranks the available plaintexts and ciphertexts separately by frequency, and maps each ciphertext to the plaintext in the same frequency rank. In this paper, we examine how frequency analysis can be used to attack encrypted deduplication.

III. Threat Model

We focus on backup workloads, which have substantial content redundancy and are proven to be effective for deduplication in practice [45], [49]. Backups are copies of primary data (e.g., application states, file systems, and virtual disk images) over time. They are typically represented as weekly full backups (i.e., complete copies of data) followed by daily incremental backups (i.e., changes of data since the last full backup). Our threat model focuses on comparing different versions of full backups from the same primary data source at different times. In the following discussion, we simply refer to "full backups" as "backups".

We consider an adversary that launches frequency analysis against an encrypted deduplication storage system that applies MLE-based deterministic encryption (e.g., convergent encryption [16] and server-aided MLE [8]) to each chunk of a backup. We assume that the adversary is honest-but-curious, meaning that it does not change the prescribed protocols of the storage system and modify any data in storage.

To launch frequency analysis, the adversary should have access to *auxiliary information* [36] that provides ground truths about the backups being stored. In this work, we model the auxiliary information as the *plaintext chunks of a prior (non-latest) backup*, which may be obtained through unintended data releases [6] or data breaches [20]. Clearly, the success of frequency analysis heavily depends on how accurate the available auxiliary information describes the backups [36]. Our focus is *not* to address how to obtain accurate auxiliary information, which we pose as future work; instead, given the available auxiliary information, we study how an adversary can design a severe attack based on frequency analysis and how we can defend against the attack. We also evaluate the attack given publicly available auxiliary information (see Section V).

Based on the available auxiliary information (which describes a prior backup), the primary goal of the adversary is to *infer* the content of the plaintext chunks that are mapped to the ciphertext chunks of the latest backup. The attack can be based on two modes:

- *Ciphertext-only mode:* It models a typical case in which the adversary can access the ciphertext chunks of the latest backup (as well as the auxiliary information about a prior backup).
- *Known-plaintext mode:* It models a more severe case in which a powerful adversary not only can access the ciphertext chunks of the latest backup and the auxiliary information about a prior backup as in ciphertext-only mode, but also knows a small fraction of the ciphertext-plaintext chunk pairs about the latest backup (e.g., from stolen devices [15]).

In both attack modes, we assume that the adversary can monitor the processing sequence of the storage system and access the *logical order* of ciphertext chunks of the latest backup before deduplication. Our rationale is that existing deduplication storage systems [47], [49] often process chunks in logical order, so as to effectively cache metadata for efficient deduplication. On the other hand, the adversary cannot access any metadata information (e.g., the fingerprint index, file recipes of all files). In practice, we do not apply deduplication to the metadata, which can be protected by conventional encryption. For example, the file recipes can be encrypted by user-specific secret keys. Also, the adversary cannot identify which prior backup a stored ciphertext chunk belongs to by analyzing the physical storage space, as the storage system can store ciphertext chunks in randomized physical addresses or commercial public clouds (the latter is more difficult to access directly).

While this work focuses on frequency analysis, another inference attack based on combinatorial optimization, called l_p-optimization, has been proposed to attack deterministic encryption [36]. Nevertheless, frequency analysis is shown to be as effective as the l_p-optimization attack in experiments [36], and later studies [27], [37] even point out that both frequency analysis and l_p-optimization may have equivalent severity. Thus, we believe that frequency analysis is representative as our baseline.

We do not consider other threats launched against encrypted deduplication, as they can be addressed independently by existing approaches. For example, the side-channel attack against encrypted deduplication [21], [22] can be addressed by server-side deduplication [22], [28] and proof of ownership [21]; the leakage of access pattern [23] can be addressed by oblivious RAM [42] and blind storage [35].

IV. Attacks

In this section, we present inference attacks based on frequency analysis against encrypted deduplication. We first present a basic attack (see Section IV-A), which builds on classical frequency analysis to infer plaintext content in encrypted deduplication. We next propose a more severe locality-based attack (see Section IV-B), which enhances the basic attack by exploiting chunk locality.

Table I summarizes the major notation used in this paper. We first formalize the adversarial goal of both basic and locality-based attacks based on the threat model in Section III. Let $\mathbf{C} = \langle C_1, C_2, \ldots \rangle$ be the sequence of ciphertext chunks in logical order for the latest backup, and $\mathbf{M} = \langle M_1, M_2, \ldots \rangle$ be the sequence of plaintext chunks in logical order for a prior backup (i.e., $\mathbf{M}$ is the auxiliary information). Both $\mathbf{C}$ and $\mathbf{M}$ show the logical orders of chunks before deduplication as perceived by the adversary (i.e., identical chunks may repeat), and each of them can have multiple identical chunks that have the same content. Note that both $\mathbf{C}$ and $\mathbf{M}$ do not necessarily have the same number of chunks. Furthermore, the i-th plaintext chunk M_i in $\mathbf{M}$ (where $i \geq 1$) is not necessarily mapped to the i-th ciphertext chunk in $\mathbf{C}$; in fact, M_i may not be mapped to any ciphertext chunk in $\mathbf{C}$, for example, when M_i has been updated before the latest backup is generated. Given $\mathbf{C}$ and $\mathbf{M}$, the goal of an adversary is to infer the content of the original plaintext chunks in $\mathbf{C}$.

A. Basic Attack

We first demonstrate how we can apply frequency analysis to infer the original plaintext chunks of the latest backup in encrypted deduplication. We call this attack the *basic attack*.

Overview: In the basic attack, we identify each chunk by its fingerprint, and count the frequency of each chunk by the number of fingerprints that appear in a backup. Thus, a chunk (or a fingerprint) has a high frequency if there exist many identical chunks with the same content. We sort the chunks of both $\mathbf{C}$ and $\mathbf{M}$ by their frequencies. We then infer that the i-th frequent plaintext chunk in $\mathbf{M}$ is the original plaintext chunk of the i-th frequent ciphertext chunk in $\mathbf{C}$. Our rationale is that the frequency of a plaintext chunk is correlated to that of a ciphertext chunk due to deterministic encryption.

TABLE I
MAJOR NOTATION USED IN THIS PAPER.

Notation	Description
Defined in Section IV	
$\mathbf{C}$	sequence of ciphertext chunks $\langle C_1, \ldots \rangle$ in logical order for the latest backup
$\mathbf{M}$	sequence of plaintext chunks $\langle M_1, \ldots \rangle$ in logical order for a prior backup
$\mathbf{F_C}$	associative array that maps each ciphertext chunk in $\mathbf{C}$ to its frequency
$\mathbf{F_M}$	associative array that maps each plaintext chunk in $\mathbf{M}$ to its frequency
$\mathcal{T}$	set of inferred ciphertext-plaintext chunk pairs
$\mathcal{L}_C$	set of left neighbors of ciphertext chunk C
$\mathcal{L}_M$	set of left neighbors of plaintext chunk M
$\mathcal{R}_C$	set of right neighbors of ciphertext chunk C
$\mathcal{R}_M$	set of right neighbors of plaintext chunk M
$\mathcal{G}$	set of currently inferred ciphertext-plaintext chunk pairs
u	number of ciphertext-plaintext chunk pairs returned from frequency analysis during the initialization of $\mathcal{G}$
v	number of ciphertext-plaintext chunk pairs returned from frequency analysis in each iteration of locality-based attack
w	maximum size of $\mathcal{G}$
$\mathbf{L_C}$	associative array that maps each ciphertext chunk in $\mathbf{C}$ to its left neighbor and co-occurrence frequency
$\mathbf{L_M}$	associative array that maps each plaintext chunk in $\mathbf{M}$ to its left neighbor and co-occurrence frequency
$\mathbf{R_C}$	associative array that maps each ciphertext chunk in $\mathbf{C}$ to its right neighbor and co-occurrence frequency
$\mathbf{R_M}$	associative array that maps each plaintext chunk in $\mathbf{M}$ to its right neighbor and co-occurrence frequency
Defined in Section VI	
K_S	segment-based key of segment S
h	minimum fingerprint of chunks in a segment

Algorithm details: Algorithm 1 shows the pseudo-code of the basic attack. It takes $\mathbf{C}$ and $\mathbf{M}$ as input, and returns the result set $\mathcal{T}$ of all inferred ciphertext-plaintext chunk pairs. It first calls the function COUNT to obtain the frequencies of all ciphertext and plaintext chunks, identified by fingerprints, in associative arrays $\mathbf{F_C}$ and $\mathbf{F_M}$, respectively (Lines 2-3). It then calls the function FREQ-ANALYSIS to infer the set $\mathcal{T}$ of ciphertext-plaintext chunk pairs (Line 4), and returns $\mathcal{T}$ (Line 5).

The function COUNT constructs an associative array $\mathbf{F_X}$ (where $\mathbf{X}$ can be either $\mathbf{C}$ and $\mathbf{M}$) that holds the frequencies of all chunks. If a chunk X does not exist in $\mathbf{F_X}$ (i.e., its fingerprint is not found), then the function adds X to $\mathbf{F_X}$ and initializes $\mathbf{F_X}[X]$ as zero (Lines 10-12). The function then increments $\mathbf{F_X}[X]$ by one (Line 13).

The function FREQ-ANALYSIS performs frequency analysis based on $\mathbf{F_C}$ and $\mathbf{F_M}$. It first sorts each of $\mathbf{F_C}$ and $\mathbf{F_M}$ by frequency (Lines 18-19). Since $\mathbf{F_C}$ and $\mathbf{F_M}$ may not have the same number of elements, it finds the minimum number of elements in $\mathbf{F_C}$ and $\mathbf{F_M}$ (Line 20). Finally, it returns the ciphertext-plaintext chunk pairs, in which both the ciphertext and plaintext chunks of each pair have the same rank (Lines 21-26).

Algorithm 1 Basic Attack

```
procedure BASIC ATTACK($\mathbf{C}, \mathbf{M}$)
    $\mathbf{F_C} \leftarrow$ COUNT($\mathbf{C}$)
    $\mathbf{F_M} \leftarrow$ COUNT($\mathbf{M}$)
    $\mathcal{T} \leftarrow$ FREQ-ANALYSIS($\mathbf{F_C}, \mathbf{F_M}$)
    return $\mathcal{T}$
end procedure

function COUNT($\mathbf{X}$)
    Initialize $\mathbf{F_X}$
    for each $X$ in $\mathbf{X}$ do
        if $X$ does not exist in $\mathbf{F_X}$ then
            Initialize $\mathbf{F_X}[X] \leftarrow 0$
        end if
        $\mathbf{F_X}[X] \leftarrow \mathbf{F_X}[X] + 1$
    end for
    return $\mathbf{F_X}$
end function

function FREQ-ANALYSIS($\mathbf{F_C}, \mathbf{F_M}$)
    Sort $\mathbf{F_C}$ by frequency
    Sort $\mathbf{F_M}$ by frequency
    $min \leftarrow \min\{|\mathbf{F_C}|, |\mathbf{F_M}|\}$
    for $i = 1$ to $min$ do
        $C \leftarrow$ $i$-th frequent ciphertext chunk
        $M \leftarrow$ $i$-th frequent plaintext chunk
        Add $(C, M)$ to $\mathcal{T}'$
    end for
    return $\mathcal{T}'$
end function
```

Discussion: The basic attack demonstrates how frequency analysis can be applied to encrypted deduplication. However, it only achieves small inference accuracy, as shown in our trace-driven evaluation (see Section V). One reason is that the basic attack is sensitive to data updates that occur across different versions of backups over time. An update to a chunk can change the frequency ranks of multiple chunks, including the chunk itself and other chunks with similar frequencies. Another reason is that there exist many ties, in which chunks have the same frequency. How to break a tie during sorting also affects the frequency rank and hence the inference accuracy of the tied chunks. In the following, we extend the basic attack to improve its inference accuracy.

B. Locality-based Attack

We propose the *locality-based attack*, which exploits *chunk locality* [30], [47], [49] to improve the severity of frequency analysis.

Overview: We first define the notation that captures the notion of chunk locality. Consider two ordered pairs $\langle C_i, C_{i+1}\rangle$ and $\langle M_i, M_{i+1}\rangle$ of neighboring ciphertext and plaintext chunks in $\mathbf{C}$ and $\mathbf{M}$, respectively. We say that C_i is the *left* neighbor of C_{i+1}, while C_{i+1} is the *right* neighbor of C_i; similar definitions apply to M_i and M_{i+1}. Note that a ciphertext chunk in $\mathbf{C}$ or a plaintext chunk in $\mathbf{M}$ may repeat many times (i.e., there are many duplicate copies), so if we identify each chunk by its fingerprint, it can be associated with more than one left or right neighbor. Let $\mathcal{L}_C$ and $\mathcal{R}_C$ be the sets of left neighbors and right neighbors of a ciphertext chunk C, respectively, and $\mathcal{L}_M$ and $\mathcal{R}_M$ be the left and right neighbors of a plaintext chunk M, respectively.

Our insight is that if a plaintext chunk M of a prior backup has been identified as the original plaintext chunk of a ciphertext chunk C of the latest backup, then the left and right neighbors of M are also likely to be original plaintext chunks of the left and right neighbors of C, mainly because chunk locality implies that the ordering of chunks is likely to be preserved across backups. In other words, for any inferred ciphertext-plaintext chunk pair (C, M), we further infer more ciphertext-plaintext chunk pairs through the left and right neighboring chunks of C and M, and repeat the same inference on those newly inferred chunk pairs. Thus, we can significantly increase the attack severity.

The locality-based attack operates on an *inferred set* $\mathcal{G}$, which stores the currently inferred set of ciphertext-plaintext chunks pairs. How to initialize $\mathcal{G}$ depends on the attack modes (see Section III). In ciphertext-only mode, in which an adversary only knows $\mathbf{C}$ and $\mathbf{M}$, we apply frequency analysis to find the most frequent ciphertext-plaintext chunk pairs and add them to $\mathcal{G}$. Here, we configure a parameter u to indicate the number of most frequent chunk pairs to be returned (e.g., $u = 5$ by default in our implementation). Our rationale is that a small number of top-frequent chunks often have significantly high frequencies (see Figure 1), and their frequency ranks are relatively stable across backups. This ensures the correctness of the ciphertext-plaintext chunk pairs in $\mathcal{G}$ with a high probability throughout the attack. On the other hand, in known-plaintext mode, in which the adversary knows some leaked ciphertext-plaintext chunk pairs about $\mathbf{C}$ for the latest backup, we initialize $\mathcal{G}$ with the set of leaked chunk pairs.

The locality-based attack proceeds as follows. In each iteration, it picks one ciphertext-plaintext chunk pair (C, M) from $\mathcal{G}$. It collects the corresponding sets of neighboring chunks $\mathcal{L}_C$, $\mathcal{L}_M$, $\mathcal{R}_C$, and $\mathcal{R}_M$. We apply frequency analysis to find the most frequent ciphertext-plaintext chunk pairs from each of $\mathcal{L}_C$ and $\mathcal{L}_M$, and similarly from $\mathcal{R}_C$ and $\mathcal{R}_M$. In other words, we find the left and right neighboring chunks of C and M that have the most *co-occurrences* with C and M themselves, respectively. We configure a parameter v (e.g., $v = 30$ by default in our implementation) to indicate the number of most frequent chunk pairs returned from frequent analysis in an iteration. A larger v increases the number of inferred ciphertext-plaintext chunk pairs, but it also potentially compromises the inference accuracy. The attack adds all inferred chunk pairs into $\mathcal{G}$, and iterates until all inferred chunk pairs in $\mathcal{G}$ have been processed.

Note that $\mathcal{G}$ may grow very large as the backup size increases. A very large $\mathcal{G}$ can exhaust memory space. We configure a parameter w (e.g., w = 200,000 by default in our implementation) to bound the maximum size of $\mathcal{G}$.

In our evaluation (see Section V), we carefully examine the impact of the configurable parameters u, v, and w.

Algorithm details: Algorithm 2 shows the pseudo-code of the locality-based attack. It takes $\mathbf{C}$, $\mathbf{M}$, u, v, and w as input,

Algorithm 2 Locality-based Attack

```
1: procedure Locality-based attack($\mathbf{C}, \mathbf{M}, u, v, w$)
2:     $(\mathbf{F_C}, \mathbf{L_C}, \mathbf{R_C}) \leftarrow$ Count($\mathbf{C}$)
3:     $(\mathbf{F_M}, \mathbf{L_M}, \mathbf{R_M}) \leftarrow$ Count($\mathbf{M}$)
4:     if ciphertext-only mode then
5:         $\mathcal{G} \leftarrow$ Freq-analysis($\mathbf{F_C}, \mathbf{F_M}, u$)
6:     else if known-plaintext mode then
7:         $\mathcal{G} \leftarrow$ set of leaked ciphertext-plaintext chunk pairs
8:     end if
9:     $\mathcal{T} \leftarrow \mathcal{G}$
10:    while $\mathcal{G}$ is non-empty do
11:        Remove $(C, M)$ from $\mathcal{G}$
12:        $\mathcal{T}_l \leftarrow$ Freq-analysis($\mathbf{L_C}[C], \mathbf{L_M}[M], v$)
13:        $\mathcal{T}_r \leftarrow$ Freq-analysis($\mathbf{R_C}[C], \mathbf{R_M}[M], v$)
14:        for each $(C, M)$ in $\mathcal{T}_l \cup \mathcal{T}_r$ do
15:            if $(C, *)$ is not in $\mathcal{T}$ then
16:                Add $(C, M)$ to $\mathcal{T}$
17:                if $|\mathcal{G}| \leq w$ (i.e., $\mathcal{G}$ is not full) then
18:                    Add $(C, M)$ to $\mathcal{G}$
19:                end if
20:            end if
21:        end for
22:    end while
23:    return $\mathcal{T}$
24: end procedure

25: function Count($\mathbf{X}$)
26:    Initialize $\mathbf{F_X}$, $\mathbf{L_X}$, and $\mathbf{R_X}$
27:    for each $X$ in $\mathbf{X}$ do
28:        if $X$ does not exist in $\mathbf{F_X}$ then
29:            Initialize $\mathbf{F_X}[X] \leftarrow 0$
30:        end if
31:        $\mathbf{F_X}[X] \leftarrow \mathbf{F_X}[X] + 1$
32:        if $X$ has a left neighbor $X_l$ then
33:            if $X_l$ does not exist in $\mathbf{L_X}[X]$ then
34:                Initialize $\mathbf{L_X}[X][X_l] \leftarrow 0$
35:            end if
36:            $\mathbf{L_X}[X][X_l] \leftarrow \mathbf{L_X}[X][X_l] + 1$
37:        end if
38:        if $X$ has a right neighbor $X_r$ then
39:            if $X_r$ does not exist in $\mathbf{R_X}[X]$ then
40:                Initialize $\mathbf{R_X}[X][X_r] \leftarrow 0$
41:            end if
42:            $\mathbf{R_X}[X][X_r] \leftarrow \mathbf{R_X}[X][X_r] + 1$
43:        end if
44:    end for
45:    return $(\mathbf{F_X}, \mathbf{L_X}, \mathbf{R_X})$
46: end function

47: function Freq-analysis($\mathbf{Y_C}$, $\mathbf{Y_M}, x$)
48:    Sort $\mathbf{Y_C}$ by frequency
49:    Sort $\mathbf{Y_M}$ by frequency
50:    for $i = 1$ to $x$ do
51:        $C \leftarrow i$-th frequent ciphertext chunk
52:        $M \leftarrow i$-th frequent plaintext chunk
53:        Add $(C, M)$ to $\mathcal{T}'$
54:    end for
55:    return $\mathcal{T}'$
56: end function
```

and returns the result set $\mathcal{T}$ of all inferred ciphertext-plaintext chunk pairs. It first calls the function Count to obtain the following associative arrays: $\mathbf{F_C}$, which stores the frequencies of all ciphertext chunks, as well as $\mathbf{L_C}$ and $\mathbf{R_C}$, which store the co-occurrence frequencies of the left and right neighbors of all ciphertext chunks, respectively (Line 2); similarly, it obtains the associative arrays $\mathbf{F_M}$, $\mathbf{L_M}$, and $\mathbf{R_M}$ for the plaintext chunks (Line 3). It then initializes the inferred set $\mathcal{G}$, either by obtaining u most frequent ciphertext-plaintext chunk pairs from frequency analysis in ciphertext-only mode, or by adding the set of leaked ciphertext-plaintext chunk pairs from the latest backup in known-plaintext mode (Lines 4-8). It also initializes $\mathcal{T}$ with $\mathcal{G}$ (Line 9).

In the main loop (Lines 10-22), the algorithm removes a pair (C, M) from $\mathcal{G}$ (Line 11) and uses it to infer additional ciphertext-plaintext chunk pairs from the neighboring chunks of C and M. It first examines all left neighbors by running the function Freq-analysis on $\mathbf{L_C}[C]$ and $\mathbf{L_M}[M]$, and stores v most frequent ciphertext-plaintext chunk pairs in $\mathcal{T}_l$ (Line 12). Similarly, it examines all right neighbors and stores the results in $\mathcal{T}_r$ (Line 13). For each (C, M) in $\mathcal{T}_l \cup \mathcal{T}_r$, if $(C, *)$ is not in $\mathcal{T}$ (i.e., the ciphertext chunk C has not been inferred yet), we add (C, M) to $\mathcal{T}$ and also to $\mathcal{G}$ if $\mathcal{G}$ is not full (Lines 14-21). The main loop iterates until $\mathcal{G}$ becomes empty. Finally, $\mathcal{T}$ is returned.

Both the functions Count and Freq-analysis are similar to those in the basic attack (see Algorithm 1), with the following extensions. For Count, in addition to constructing the associative array $\mathbf{F_X}$ (where $\mathbf{X}$ can be either $\mathbf{C}$ and $\mathbf{M}$) that holds the frequencies of all chunks, it also constructs the associative arrays $\mathbf{L_X}$ and $\mathbf{R_X}$ that hold the co-occurrence frequencies of the left and right neighbors of each chunk X, respectively. For Freq-analysis, it now performs frequency analysis on the associative arrays $\mathbf{Y_C}$ and $\mathbf{Y_M}$, in which $\mathbf{Y_C}$ (resp. $\mathbf{Y_M}$) refers to either $\mathbf{F_C}$ (resp. $\mathbf{F_M}$) that holds the frequency counts of all chunks, or $\mathbf{L_C}[C]$ and $\mathbf{R_C}[C]$ (resp. $\mathbf{L_M}[M]$ and $\mathbf{R_M}[M]$) that hold the frequency counts of all ordered pairs of chunks associated with ciphertext chunk C (resp. plaintext chunk M). Also, Freq-analysis only returns x (where x can be either u or v) most frequent ciphertext-plaintext chunk pairs.

Example: Figure 2 shows an example of how the locality-based attack works. Here, we consider ciphertext-only mode. Suppose that we have obtained the auxiliary information $\mathbf{M} = \langle M_1, M_2, M_1, M_2, M_3, M_4, M_2, M_3, M_4 \rangle$ of some prior backup, and use it to infer the original plaintext chunks of $\mathbf{C} = \langle C_1, C_2, C_5, C_2, C_1, C_2, C_3, C_4, C_2, C_3, C_4, C_4 \rangle$ of the latest backup. We set $u = v = 1$, and $w \to \infty$ (i.e., the inferred set $\mathcal{G}$ is unbounded). We assume that the ground truth is that the original plaintext chunk of the ciphertext chunk C_i is M_i for $i = 1, 2, 3, 4$, while that of C_5 is some new plaintext chunk not in $\mathbf{M}$ (note that in reality, an adversary does not know the ground truth).

We first apply frequency analysis and find that (C_2, M_2) is the most frequent ciphertext-plaintext chunk pair, so we initialize $\mathcal{G} = \{(C_2, M_2)\}$ and add it into $\mathcal{T}$. We then remove and operate on (C_2, M_2) from $\mathcal{G}$, and find that $\mathcal{L}_{C_2} = \{C_1, C_4, C_5\}$, $\mathcal{L}_{M_2} = \{M_1, M_4\}$, $\mathcal{R}_{C_2} = \{C_1, C_3, C_5\}$, and $\mathcal{R}_{M_2} = \{M_1, M_3\}$. From $\mathcal{L}_{C_2}$ and $\mathcal{L}_{M_2}$, we find that

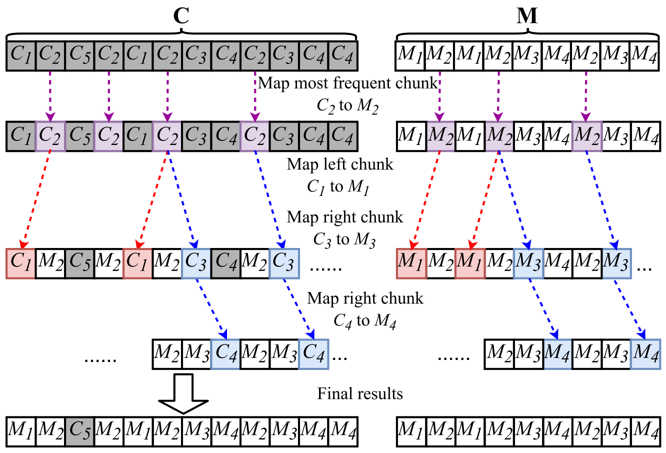

C
M
Map most frequent chunk C_2 to M_2
Map left chunk C_1 to M_1
Map right chunk C_3 to M_3
Map right chunk C_4 to M_4
Final results

Fig. 2. Example of the locality-based attack.

(C_1, M_1) is the most frequent ciphertext-plaintext chunk pair, while from $\mathcal{R}_{C_2}$ and $\mathcal{R}_{M_2}$, we find (C_3, M_3). Thus, we add both (C_1, M_1) and (C_3, M_3) into $\mathcal{G}$ and $\mathcal{T}$. We repeat the processing on (C_1, M_1) and (C_3, M_3), and we can infer another pair (C_4, M_4) from the right neighbors of (C_3, M_3).

To summarize, the locality-based attack can successfully infer the original plaintext chunks of all four ciphertext chunks C_1, C_2, C_3, and C_4. It cannot infer the original plaintext chunk of C_5, as it does not appear in $\mathbf{M}$.

V. Attack Evaluation

In this section, we present trace-driven evaluation results to show the severity of frequency analysis against encrypted deduplication.

A. Implementation

We implement both the basic and locality-based attacks in C++. We benchmark our current implementation on a Ubuntu 16.04 Linux machine with an AMD Athlon II X4 640 quad-core 3.0GHz CPU and 16GB RAM, and find that the locality-based attack takes around 15 hours to process an FSL backup of size around 500GB (see Section V-B for dataset details). In the following, we highlight the implementation details of some data structures used by the attacks.

Associative arrays: Recall that there are three types of associative arrays: (i) $\mathbf{F_C}$ and $\mathbf{F_M}$, (ii) $\mathbf{L_C}$ and $\mathbf{L_M}$, and (iii) $\mathbf{R_C}$ and $\mathbf{R_M}$ (the latter two are only used by the locality-based attack). We implement them as key-value stores using LevelDB [19]. Each key-value store is keyed by the fingerprint of the ciphertext/plaintext chunk. For $\mathbf{F_C}$ and $\mathbf{F_M}$, each entry stores a frequency count; for $\mathbf{L_C}$, $\mathbf{L_M}$, $\mathbf{R_C}$, and $\mathbf{R_M}$, each entry stores a sequential list of the fingerprints of all the left/right neighbors of the keyed chunk and the co-occurrence frequency counts. For the latter, keeping neighboring chunks sequentially simplifies our implementation, but also increases the search time of a particular neighbor (which dominates the overall running time); we pose the optimization as future work.

Inferred set: We implement the inferred set $\mathcal{G}$ in the locality-based attack as a first-in-first-out queue, whose maximum size is bounded by w (see Section IV-B). Each time we remove the first ciphertext-plaintext chunk pair from the queue for inferring more chunk pairs from the neighbors.

B. Datasets

We consider two types of datasets to drive our evaluation.

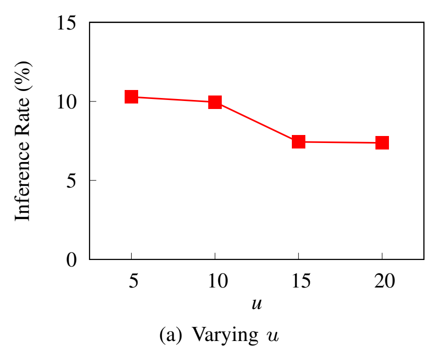
FSL: This is a real-world dataset collected by the File systems and Storage Lab (FSL) at Stony Brook University [2], [44]. We focus on the *Fslhomes* dataset, which contains the daily snapshots of users' home directories on a shared file system. Each snapshot is represented by a collection of 48-bit chunk fingerprints produced by variable-size chunking of different average sizes. We pick the snapshots from January 22 to May 21 in 2013, and fix the average size as 8KB for our evaluation. We select six users (User4, User7, User12, User13, User15, and User28) that have the complete daily snapshots over the whole duration. We aggregate each user's snapshots on a monthly basis (on January 22, February 22, March 22, April 21, and May 21), and hence form five monthly full backups per user. Our post-processed dataset covers a total of 2.69TB of logical data before deduplication.

Synthetic: This dataset contains a sequence of synthetic backup snapshots that are generated based on Lillibridge *et al.*'s approach [29]. Specifically, we create an initial snapshot from a Ubuntu 14.04 virtual disk image (originally with 1.1GB of data) with a total of 4.28GB space. We create a sequence of snapshots starting from the initial snapshot, such that each snapshot is created from the previous one by randomly picking 2% of files and modifying 2.5% of their content, and also adding 10MB of new data. Finally, we generate a sequence of ten snapshots, each of which is treated as a backup. Based on our choices of parameters, the resulting storage saving is around 90% (see Experiment B.2 in Section VII-B); equivalently, the deduplication ratio is around 10:1, which is typical in real-life backup workloads [45]. Note that the initial snapshot is publicly available. Later in our evaluation, we study the effectiveness of the attacks by using it as public auxiliary information.

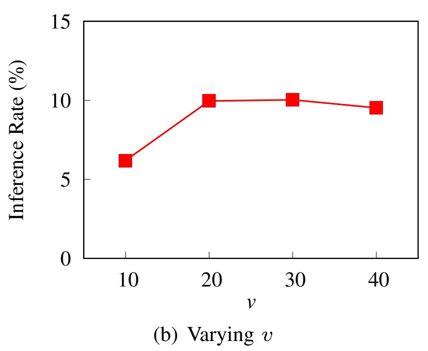
C. Results

We present evaluation results. We quantify the severity of an attack using the *inference rate*, defined as the ratio of the number of unique ciphertext chunks whose plaintext chunks are successfully inferred over the total number of unique ciphertext chunks in the latest backup; a higher inference rate implies that the attack is more severe.

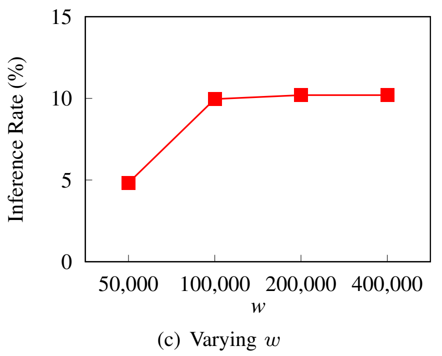
Experiment A.1 (Impact of parameters): We first evaluate the impact of parameters on the locality-based attack, in order to justify our choices of parameters. Recall that the locality-based attack is configured with three parameters: u, v, and w. In this experiment, we use the FSL dataset, and focus on the attack in ciphertext-only mode. We use the middle version of the backup on March 22 as auxiliary information, and launch the attack to infer the original plaintext chunks of the latest backup on May 21. In each experiment, we fix two out of the



(a) Varying u

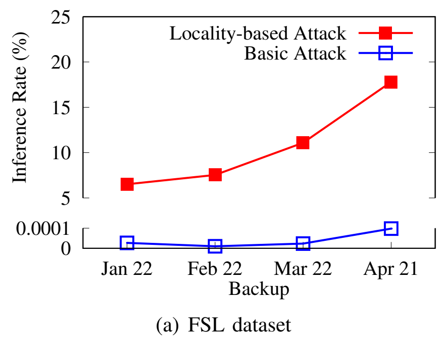


(b) Varying v

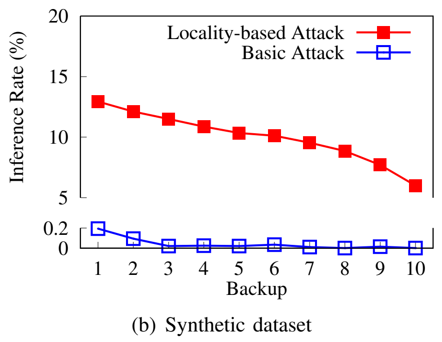


(c) Varying w

Fig. 3. Experiment A.1 (Impact of parameters).



(a) FSL dataset



(b) Synthetic dataset

Fig. 4. Experiment A.2 (Inference rate in ciphertext-only mode).

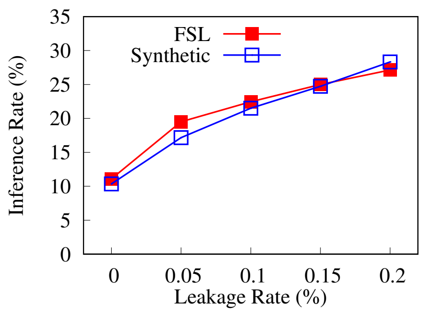


Fig. 5. Experiment A.3 (Inference rate in known-plaintext mode).

three parameters and vary the remaining parameter to identify the largest inference rate in each case.

Figure 3(a) first shows the impact of u, in which we fix $v = 20$ and $w = 100{,}000$. The inference rate decreases with u. For example, when $u = 5$, the attack can successfully infer 10.28% of plaintext chunks, while the inference rate drops to 7.37% when u increases to 20. The reason is that a larger u implies that incorrect ciphertext-plaintext chunk pairs are more likely to be included into the inferred set during initialization, thereby compromising the inference accuracy.

Figure 3(b) next shows the impact of v, in which we fix $u = 10$ and $w = 100{,}000$. Initially, the inference rate increases with v as the underlying frequency analysis infers more ciphertext-plaintext chunk pairs in each iteration. It hits the maximum value at about 10% when $v = 30$. When v increases to 40, the inference rate drops slightly to about 9.52%. The reason is that some incorrectly inferred ciphertext-plaintext chunk pairs are also included into $\mathcal{G}$, which compromises the inference rate.

Figure 3(c) finally shows the impact of w, in which we fix $u = 10$ and $v = 20$. A larger w increases the inference rate, since $\mathcal{G}$ can hold more ciphertext-plaintext chunk pairs across iterations. We observe that when w increases beyond 200,000, the inference rate becomes steady at about 10.2%.

Experiment A.2 (Inference rate in ciphertext-only mode): We now compare both basic and locality-based attacks in ciphertext-only mode. From Experiment A.1, we select $u = 5$, $v = 30$, and $w = 200{,}000$ as default parameters, as they give the highest possible inference rate for the locality-based attack.

We first consider the FSL dataset. We choose each of the prior FSL backups on January 22, February 22, March 22, and April 21 as auxiliary information, and we launch attacks to infer the original plaintext chunks in the latest backup on May 21. Figure 4 shows the inference rate versus the prior backup. As expected, the inference rate of both attacks increases as we use more recent non-latest backups as auxiliary information, since a more recent backup has higher content redundancy with the target latest backup. We observe that the basic attack is ineffective in all cases, as the inference rate is no more than 0.0001%. On the other hand, the locality-based attack can achieve a significantly high inference rate. For example, if we use the most recent non-latest backup on April 21 as auxiliary information, the inference rate of the locality-based attack can reach as high as 17.8%.

We now consider the synthetic dataset. We use the initial snapshot (which is publicly available) as auxiliary information. We then infer the original plaintext chunks in each of the following synthetic backups. Figure 4(b) shows the inference rates of both the basic and locality-based attacks over the sequence of backups. The locality-based attack is again more severe than the basic attack, whose inference rate is less than 0.2%. For example, the locality-based attack can infer 12.93% of original plaintext chunks in the first backup, while that of the basic attack is only 0.19%. After ten backups, since more chunks have been updated since the initial snapshot, the inference rates of the locality-based and basic attacks drop to 6% and 0.0007%, respectively. Nevertheless, we observe that the locality-based attack always incurs a higher inference rate than the basic attack.

Experiment A.3 (Inference rate in known-plaintext mode): We further evaluate the severity of the locality-based attack in known-plaintext mode. To quantify the amount of leakage about the latest backup (see Section III), we define the *leakage rate* as the ratio of the number of ciphertext-plaintext chunk pairs known by the adversary to the total number of ciphertext chunks in the latest backup. We consider the medium case of the attack: for the FSL dataset, we choose the middle version of the backup on March 22 as auxiliary information to infer the latest backup on May 21; for the synthetic dataset, we use the initial snapshot as auxiliary information to infer the 5th backup snapshot. We configure $u = 5$, $v = 30$, and $w =$ 500,000. Note that we increase w to 500,000 (as compared to $w =$ 200,000 in Experiment A.2), since we find that the attack in known-plaintext mode can infer much more ciphertext-plaintext chunk pairs across iterations. Thus, we choose a larger w to include them into the inferred set.

Figure 5 shows the inference rate versus the leakage rate about the target backup being inferred, which we vary from 0% to 0.2%. The slight increase in the leakage rate can lead to a significant increase in the inference rate. For example, when the leakage rate increases from 0 to 0.2%, the inference rate increases from 11.09% to 27.14% and from 10.34% to 28.32% for the FSL and synthetic datasets, respectively.

VI. Defense

The deterministic nature of encrypted deduplication discloses the frequency distribution of the underlying plaintext chunks, thereby making frequency analysis feasible. To defend against frequency analysis, we consider a *MinHash encryption* scheme that encrypts each copy of identical plaintext chunks into possibly different ciphertext chunks, so as to hide the frequency distribution of original chunks, while ensuring that the storage efficiency is only slightly degraded.

Overview: MinHash encryption builds on Broder's theorem [12], which quantifies the similarity of two sets of elements:

Broder's theorem [12]: Consider two sets of elements S_1 and S_2. Let $U = |S_1 \cup S_2|$ be the number of elements in the union of S_1 and S_2, H be a hash function that is chosen uniformly at random from a min-wise independent family of permutations, and $\min\{H(S)\}$ be the minimum element hash of S. Then $\Pr[\min\{H(S_1)\} = \min\{H(S_2)\}] = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$.

Broder's theorem states that if two sets share a large fraction of common elements (i.e., they are highly similar), then the probability that both sets share the same minimum hash element is also high. Since two backups from the same data source are expected to be highly similar and share a large number of identical chunks [45], MinHash encryption leverages this property to perform encrypted deduplication in a different way from the original MLE [8], [9]. We emphasize that previous deduplication approaches also leverage Broder's theorem to minimize the memory usage of the fingerprint index in plaintext deduplication [10], [47] or key generation overhead in server-aided MLE [38]. Thus, we do not claim the novelty of the design of MinHash encryption. Instead, our contribution is to demonstrate that it can effectively defend against frequency analysis.

Algorithm 3 MinHash Encryption

```
1:  procedure MinHash Encryption($\mathbf{M}$)
2:      Initialize $\mathbf{C}$
3:      Partition $\mathbf{M}$ into segments
4:      for each segment $S$ do
5:          $h \leftarrow$ minimum fingerprint of all chunks in $S$
6:          $K_S \leftarrow$ segment-based key derived from $h$
7:          for each chunk $M \in S$ do
8:              $C \leftarrow$Encrypt$(K_S, M)$
9:              Add $C$ into $\mathbf{C}$
10:         end for
11:     end for
12:     return $\mathbf{C}$
13: end procedure
```

Algorithm details: MinHash encryption works as follows, as shown in Algorithm 3. It takes a sequence of plaintext chunks $\mathbf{M}$ as input, and returns a sequence of ciphertext chunks $\mathbf{C}$ as output. It first partitions an original stream of plaintext chunks $\mathbf{M}$ into a set of large-size data units called *segments* (Line 3), each of which is composed of a sequence of plaintext chunks. One possible way of such partitioning is to put a segment boundary at the chunk boundary if a chunk's modulo hash equals some specified pattern, so that segment boundaries align with chunk boundaries [30]; we elaborate the implementation details in Section VII-A. For each segment S, MinHash encryption computes the minimum fingerprint h of all chunks in S; here, we use the fingerprint value as the hash value of each chunk. It then derives the segment-based key K_S based on h (Line 6), for example, by querying a key manager as in DupLESS [8] to defend against brute-force attacks (see Section II-B). It then encrypts each chunk in S using K_S and adds the resulting ciphertext chunk to $\mathbf{C}$ (Lines 7-10). Finally, it returns $\mathbf{C}$ (Line 12).

Why MinHash encryption is effective: MinHash encryption preserves deduplication effectiveness. The rationale is that segments are highly similar, so their minimum fingerprints and hence the segment-based keys are likely to be the same. Also, the similar segments share a large fraction of identical plaintext chunks, which will likely be encrypted by the same segment-based keys into identical ciphertext chunks that can be deduplicated. Since identical plaintext chunks may reside in different segments that have different segment-based keys, their resulting ciphertext chunks become different and cannot be deduplicated. Thus, we can only achieve near-exact deduplication and expect a slight degradation of storage efficiency.

Nevertheless, our main observation is that such near-exact deduplication provides resilience against frequency analysis. Since an identical chunk can now be encrypted into *multiple* distinct ciphertext chunks, this breaks the deterministic nature of encrypted deduplication. More importantly, even though it affects only a small number of identical chunks, it is sufficient to disturb the overall frequency rank of ciphertext chunks and make frequency analysis ineffective.

VII. Defense Evaluation

In this section, we present trace-driven evaluation results on the effectiveness of MinHash encryption, using the same FSL and synthetic datasets in our attack evaluation (see Section V).

A. Implementation

Since the FSL dataset does not contain actual content, instead of performing actual encryption, we simulate MinHash encryption by directly operating on chunk fingerprints.

First, we need to identify segment boundaries based on chunk fingerprints. We follow the variable-size segmentation scheme in [30]. Specifically, the segmentation scheme is configured by the minimum, average, and maximum segment sizes. It places a segment boundary at the end of a chunk fingerprint if (i) the size of each segment is at least the minimum segment size, and (ii) the chunk fingerprint modulo a pre-defined divisor (which determines the average segment size) is equal to some constant (e.g., -1), or the inclusion of the chunk makes the segment size larger than the maximum segment size. In our evaluation, we vary the average segment size, and fix the minimum and maximum segment sizes as half and double of the average segment size, respectively.

We mimic the encryption process as follows. We first calculate the minimum chunk fingerprint h of each segment. We then concatenate h with each chunk fingerprint in the segment and compute the MD5 hash of the concatenation. We also truncate hash result to keep only the first 48 bits, so as to be consistent with the fingerprint size in the original FSL dataset. The truncated hash result can be viewed as the fingerprint of the ciphertext chunk. We can easily check that identical plaintext chunks under the same h will lead to identical ciphertext chunks that can be deduplicated.

B. Results

Experiment B.1 (Robustness against leakage): We first evaluate the effectiveness of MinHash encryption against the locality-based attack. We vary the average segment size as 512KB, 1MB, and 2MB (while the average chunk size is 8KB). We use the same parameter setting as in Experiment A.3 and evaluate the inference rate versus the leakage rate.

Figure 6(a) first shows the results of the FSL dataset. In ciphertext-only mode (i.e., the leakage rate is zero), MinHash encryption suppresses the inference rate to almost zero. In particular, when the segment sizes are 512KB, 1MB, and 2MB, the locality-based attack can only successfully infer 6, 6, and 3 out of around 37 million ciphertext-plaintext chunk pairs. The main reason is the frequency rank of the ciphertext chunks has been disturbed by MinHash encryption (as explained in Section VI). In known-plaintext mode (i.e., the leakage rate is greater than zero), the inference rate increases with the leakage rate, but remains very small. For example, when the leakage rate increases to 0.2%, the inference rates are only 0.41%, 0.39%, and 0.38% when the average segment sizes are 512KB, 1MB, and 2MB, respectively. We observe that a larger segment size implies a slightly smaller inference rate. We conjecture the reason is that a larger segment size implies a smaller probability that two different segments share the same minimum chunk fingerprint in this dataset. Thus, identical plaintext chunks are more likely to be encrypted into multiple distinct ciphertext chunks, and the frequency rank of ciphertext chunks is more disturbed.

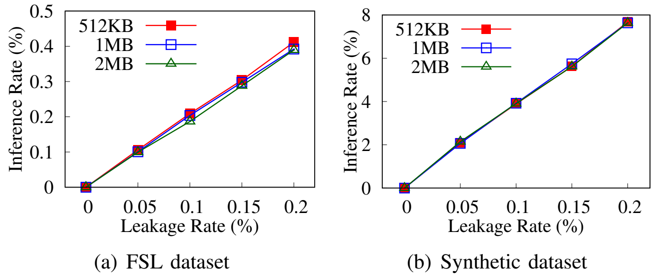
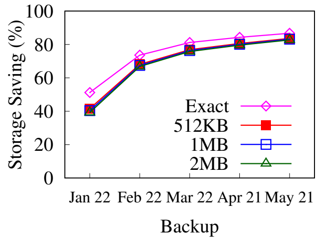


Fig. 6. Experiment B.1 (Robustness against leakage).

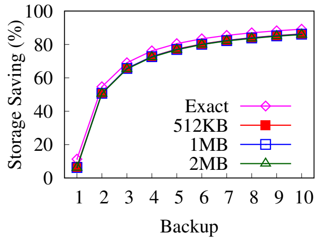
Figure 6(b) next shows the results of the synthetic dataset. In ciphertext-only mode, the locality-based attack can just successfully infer one out of about 420 thousands of ciphertext-plaintext chunk pairs for all segment sizes. Also, as in the FSL dataset, the inference rate increases with the leakage rate. However, the increasing speed of the synthetic dataset is higher than that of the FSL dataset. The reason is that each snapshot of the synthetic dataset is of a relatively small size (about 4.28GB) and forms fewer segments with different minimum fingerprints. This makes the frequency rank of ciphertext chunks less disturbed by MinHash encryption. Even so, when the leakage rate is 0.2%, the inference rate is about 7.63-7.68%, which is dramatically reduced from the previous inference rate (about 28.32%) without MinHash encryption. We pose it as a future work on how to further suppress the inference rate in small-size backups.

Experiment B.2 (Storage efficiency): Finally, we show that the storage efficiency is preserved by MinHash encryption. We add the encrypted backups to storage in the order of their creation times, and measure the *storage saving* as the percentage of the total size of all ciphertext chunks reduced by deduplication. We compare the storage saving of MinHash encryption with that of exact deduplication that operates at the chunk level and eliminates all duplicate chunks. Here, we do not consider the metadata overhead.

Figure 7(a) shows the storage saving after storing each FSL backup. We observe that after storing all five backups, the storage saving achieves 83.61%, 83.17% and 82.69% for segment sizes 512KB, 1MB, and 2MB, respectively. Compared to exact deduplication, their savings are reduced by a difference of 3.12%, 3.56%, and 4.03%, respectively. Figure 7(b) shows the storage saving after storing each synthetic snapshot (excluding the initial snapshot). After ten backups, there is a storage sav-



(a) FSL dataset



(b) Synthetic dataset

Fig. 7. Experiment B.2 (Storage efficiency).

ing of 86.28%, 86.15% and 85.88% for segment sizes 512KB, 1MB, and 2MB, respectively. The drop of the storage saving is also small (less than 4%) compared to exact deduplication, which achieves a storage saving of 89.15%. Overall, MinHash encryption maintains high storage efficiency for both datasets.

VIII. Related Work

Encrypted deduplication: Traditional encrypted deduplication systems (e.g., [4], [14], [16], [24], [43], [46]) mainly build on convergent encryption [16], in which the encryption key is directly derived from the cryptographic hash of the content to be encrypted. CDStore [28] integrates convergent encryption with secret sharing to support fault-tolerant storage. However, convergent encryption is vulnerable to brute-force attacks (see Section II-B). Server-aided MLE protects against brute-force attacks by maintaining content-to-key mappings in a dedicated key manager, and has been implemented in various storage system prototypes [5], [8], [38], [41]. Given that the dedicated key manager is a single-point-of-failure, Duan [17] proposes to maintain a quorum of key managers via threshold signature for fault-tolerant key management. Note that all the above systems build on deterministic encryption to preserve the deduplication capability of ciphertext chunks, and hence are vulnerable to the inference attacks studied in this paper.

Instead of using deterministic encryption, Bellare *et al.* [9] propose an MLE variant called *random convergent encryption (RCE)*, which uses random keys for chunk encryption. However, RCE needs to add deterministic tags into ciphertext chunks for checking any duplicates, so that the adversary can count the deterministic tags to obtain the frequency distribution. Liu *et al.* [31] propose to encrypt each plaintext chunk with a random key, while the key is shared among users via password-based key exchange. However, the proposed approach incurs significant key exchange overhead, especially when the number of chunks is huge.

From the theoretic perspective, there are several works that strengthen the security of encrypted deduplication and protect the frequency distribution of original chunks. Abadi *et al.* [3] propose two encrypted deduplication schemes for the chunks that depend on public parameters, yet either of them builds on computationally expensive non-interactive zero knowledge (NIZK) proofs or produces deterministic ciphertext components. Interactive MLE [7] addresses chunk correlation and parameter dependence, yet it is impractical for the use of fully homomorphic encryption (FHE). This paper differs from the above works by using lightweight primitives for practical encrypted deduplication.

Inference attacks: Frequency analysis [32] is the classical inference attack and has been historically used to recover plaintexts from substitution-based ciphertexts. It is also used as a building block in recently proposed attacks. Kumar *et al.* [26] use frequency-based analysis to de-anonymize query logs. Islam *et al.* [23] compromise keyword privacy based on the leakage of the access patterns in keyword search. Naveed *et al.* [36] propose to conduct frequency analysis via combinatorial optimization and present attacks against CryptDB. Kellaris *et al.* [25] propose reconstruction attacks against any system that leaks access pattern or communication volume. Pouliot *et al.* [37] present the graph matching attacks on searchable encryption. In contrast, we focus on encrypted deduplication storage and exploit workload characteristics to construct attack and defense approaches.

Ritzdorf *et al.* [40] exploit the size information of deduplicated content and build an inference attack that determines if a file has been stored. Our work is different as we focus on inferring the content of data chunks based on chunk frequencies. We further examine the effectiveness of MinHash encryption in defending our proposed attack.

Some inference attacks exploit the *active* adversarial capability. Brekne *et al.* [1] construct bogus packets to de-anonymize IP addresses. Cash *et al.* [13] and Zhang *et al.* [48] propose file-injection attacks against searchable encryption. Our proposed attacks do not rely on the active adversarial capability.

IX. Conclusion

Encrypted deduplication has been deployed in commercial cloud environments and extensively studied in the literature to simultaneously achieve both data confidentiality and storage efficiency, yet we argue that its data confidentiality remains not fully guaranteed. In this paper, we demonstrate how the deterministic nature of encrypted deduplication makes it susceptible to information leakage caused by frequency analysis. We consider a locality-based attack, which exploits the chunk locality property of backup workloads that are commonly targeted by deduplication systems, so as to infer the content of a large fraction of plaintext chunks from the ciphertext chunks of the latest backup. We show how the locality-based attack can be practically implemented, and demonstrate its severity through trace-driven evaluation on both real-world and synthetic datasets. To defend against information leakage, we consider MinHash encryption, which builds on Broder's theorem to relax the deterministic nature of encrypted deduplication by encrypting some identical plaintext chunks to multiple distinct ciphertext chunks. Our trace-driven evaluation demonstrates that MinHash encryption is robust against the locality-based attack, while maintaining deduplication effectiveness as also shown by previous deduplication approaches. The source code of our attack and defense implementations is now available at **http://adslab.cse.cuhk.edu.hk/software/freqanalysis**.

Acknowledgments

This work was supported in part by grants by National Natural Science Foundation of China (Grant No. 61602092), Fundamental Research Funds for the Central Universities (Grant No. ZYGX2016KYQD115), Key Basic Research of Sichuan Province (Grant No. 2016JY0007), the Research Grants Council of Hong Kong (GRF CUHK413813 and CRF C7036-15), and Cisco University Research Program Fund (CG#593822) from Silicon Valley Community Foundation.

References

[1] "Anonymization of IP traffic monitoring data: Attacks on two prefix-preserving anonymization schemes and some proposed remedies," in *Proc. of PETs*, 2006.

[2] "FSL traces and snapshots public archive," http://tracer.filesystems.org/, 2014.

[3] M. Abadi, D. Boneh, I. Mironov, A. Raghunathan, and G. Segev, "Message-locked encryption for lock-dependent messages," in *Proc. of CRYPTO*, 2013.

[4] P. Anderson and L. Zhang, "Fast and secure laptop backups with encrypted de-duplication," in *Proc. of USENIX LISA*, 2010.

[5] F. Armknecht, J.-M. Bohli, G. O. Karame, and F. Youssef, "Transparent data deduplication in the cloud," in *Proc. of ACM CCS*, 2015.

[6] M. Arrington, "AOL: "this was a screw up"," https://techcrunch.com/2006/08/07/aol-this-was-a-screw-up/, 2006.

[7] M. Bellare and S. Keelveedhi, "Interactive message-locked encryption and secure deduplication," in *Proc. of PKC*, 2015.

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

[9] ——, "Message-locked encryption and secure deduplication," in *Proc. of EUROCRYPT*, 2013.

[10] D. Bhagwat, K. Eshghi, D. D. Long, and M. Lillibridge, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in *Proc. of IEEE MASCOTS*, 2009.

[11] J. Black, "Compare-by-hash: a reasoned analysis," in *Proc. of USENIX ATC*, 2006.

[12] A. Z. Broder, "On the resemblance and containment of documents," in *Proc. of IEEE Compression and Complexity of Sequences*, 1997.

[13] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proc. of ACM CCS*, 2015.

[14] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: Making backup cheap and easy," in *Proc. of USENIX OSDI*, 2002.

[15] B. Darrow, "Harvard-affiliate McLean hospital loses patient data," http://fortune.com/2015/07/29/mclean-hospital-loses-patient-data/, 2015.

[16] 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*, 2002.

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

[18] K. Eshghi and H. K. Tang, "A framework for analyzing and improving content-based chunking algorithms," HPL-2005-30R1, 2005.

[19] S. Ghemawat and J. Dean, "LevelDB: A fast key/value storage library by Google," https://github.com/google/leveldb, 2014.

[20] R. Hackett, "Linkedin lost 167 million account credentials in data breach," http://fortune.com/2016/05/18/linkedin-data-breach-email-password/, 2016.

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

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

[23] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *Proc. of NDSS*, 2012.

[24] M. Kallahall, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus: Scalable secure file sharing on untrusted storage," in *Proc. of USENIX FAST*, 2002.

[25] G. Kellaris, G. Kollios, K. Nissim, and A. ONeill, "Generic attacks on secure outsourced databases," in *Proc. of ACM CCS*, 2016.

[26] R. Kumar, J. Novak, B. Pang, and A. Tomkins, "On anonymizing query logs via token-based hashing," in *Proc. of ACM WWW*, 2007.

[27] M.-S. Lacharité and K. G. Paterson, "A note on the optimality of frequency analysis vs. l_p-optimization," *IACR Cryptology ePrint Archive*, 2015.

[28] M. Li, C. Qin, and P. P. C. Lee, "CDStore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal," in *Proc. of USENIX ATC*, 2015.

[29] M. Lillibridge, K. Eshghi, and D. Bhagwat, "Improving restore speed for backup systems that use inline chunk-based deduplication," in *Proc. of USENIX FAST*, 2013.

[30] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, "Sparse indexing: Large scale, inline deduplication using sampling and locality," in *Proc. of USENIX FAST*, 2009.

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

[32] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, "Handbook of applied cryptography," 2001.

[33] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," in *Proc. of USENIX FAST*, 2011.

[34] 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 *Proc. of USENIX Security*, 2011.

[35] M. Naveed, M. Prabhakaran, and C. Gunter, "Dynamic searchable encryption via blind storage," in *Proc. of IEEE S&P*, May 2014.

[36] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *Proc. of ACM CCS*, 2015.

[37] D. Pouliot and C. V. Wright, "The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption," in *Proc. of ACM CCS*, 2016.

[38] C. Qin, J. Li, and P. P. C. Lee, "The design and implementation of a rekeying-aware encrypted deduplication storage system," *ACM Trans. on Storage*, vol. 13, no. 1, pp. 9:1–9:30, Mar 2017.

[39] M. O. Rabin, "Fingerprinting by random polynomials," Center for Research in Computing Technology, Harvard University. Tech. Report TR-CSE-03-01, 1981.

[40] H. Ritzdorf, G. Karame, C. Soriente, and S. Čapkun, "On Information Leakage in Deduplicated Storage Systems," in *Proc. of ACM CCSW*, 2016.

[41] P. Shah and W. So, "Lamassu: Storage-efficient host-side encryption," in *Proc. of USENIX ATC*, 2015.

[42] E. Shi, T.-H. H. Chan, E. Stefanov, and M. Li, "Oblivious RAM with $o((\log n)^3)$ worst-case cost," in *Proc. of ASIACRYPT*, 2011.

[43] M. W. Storer, K. Greenan, D. D. Long, and E. L. Miller, "Secure data deduplication," in *Proc. of ACM StorageSS*, 2008.

[44] Z. Sun, G. Kuenning, S. Mandal, P. Shilane, V. Tarasov, N. Xiao, and E. Zadok, "A long-term user-centric analysis of deduplication patterns," in *Proc. of IEEE MSST*, 2016.

[45] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu, "Characteristics of backup workloads in production systems," in *Proc. of USENIX FAST*, 2012.

[46] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: The least-authority filesystem," in *Proc. of ACM StorageSS*, 2008.

[47] 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 *Proc. of USENIX ATC*, 2011.

[48] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: the power of file-injection attacks on searchable encryption," in *Proc. of USENIX Security*, 2016.

[49] B. Zhu, K. Li, and R. H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *Proc. of USENIX FAST*, 2008.