电子科技大学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

本科学位论文

BACHELOR THESIS

论文题目 密文重复数据删除机制的频率攻击方法

学科专业	信息安全
学　　号	2015040101018
作者姓名	任彦璟
指导老师	李经纬　副教授

分类号 _____ 密级 _____

UDC 注 1 _____

学　位　论　文

密文重复数据删除机制的频率攻击方法

（题名和副题名）

任彦璟

（作者姓名）

指导老师 李经纬　副教授

（姓名、职称、单位名称）

申请学位级别 本科 学科专业 信息安全

提交论文日期 _____ 论文答辩日期 _____

学位授予单位和日期 电子科技大学

答辩委员会主席 _____

评阅人 _____

注 1：注明《国际十进分类法 UDC》的类号。

摘　要

加密重复数据删除旨在解决大规模数据存储系统中的安全性和存储效率：它确保每个明文都由明文本身内容派生的对称密钥加密，从而为数据存储提供机密性保证，同时保留重复数据删除的存储节省效率。但是，加密重复数据删除的确定性特性也会泄漏明文的频率，从而使加密重复数据删除容易受到频率分析的影响。在本文中，我们重新审视了频率分析导致的加密重复数据删除的安全漏洞。我们认为加密重复数据删除可能更容易受到精心设计的频率分析攻击，因此攻击提供了高可信度，可确保每个推断的明文确实对应于目标密文的概率很高（即从统计角度来看，高精度）。为此，我们提出了两种新的频率分析攻击，它们可以适应实际重复数据删除工作负载的特性，从而提高频率分析的准确性。我们根据实际情况评估针对实际存储工作负载的建议攻击，并提供有关如何带来实际损失的观察。

关键词： 加密重复数据删除, 频率分析攻击, 聚类分析

ABSTRACT

Encrypted deduplication aims to address security and storage efficiency in large-scale data storage systems: it ensures that each plaintext is encrypted by a symmetric key that is derived from the content of the plaintext itself, so as to provide confidentiality guarantees for data storage while preserving the storage saving effectiveness of deduplication. However, the deterministic nature of encrypted deduplication also leaks the frequencies of plaintexts, thereby making encrypted deduplication vulnerable to frequency analysis. In this paper, we revisit the security vulnerability of encrypted deduplication due to frequency analysis. We argue that encrypted deduplication can be even more vulnerable to carefully crafted frequency analysis attacks, such that the attacks provide high confidence of ensuring that each inferred plaintext indeed corresponds to the target ciphertext with a high probability (i.e., high precision from a statistical perspective). To this end, we propose two new frequency analysis attacks that adapt the characteristics of practical deduplication workloads to increasing the severity of frequency analysis. We empirically evaluate our proposed attacks against real-world storage workloads, and provide observations on how they bring actual damages.

Keywords: Encrypted deduplication, Frequency analysis attacks, Cluster analysis

目 录

第一章 绪 论

1.1 研究工作的背景与意义

在本节中，我们将介绍普通和加密重复数据删除方法的基础知识。我们专注于基于块的重复数据删除，它以称为块的小型数据单元的粒度运行。图 1 总结了重复数据删除工作流程。具体地，存储系统首先经由一些组块过程将客户端的文件（例如，备份）分割成逻辑组块，使得每个逻辑组块由其内容的加密哈希唯一地标识，称为标签（即，指纹）。如果两个逻辑块具有相同的标记，则认为它们是相同的，而不同逻辑块具有相同标记的可能性几乎可以忽略不计 [24]。存储系统仅存储相同逻辑块的副本作为物理块，并且每个相同的逻辑块通过小尺寸引用引用相同的物理块。基于块的重复数据删除比整个文件重复数据删除更精细，因此通常具有更高的存储效率。我们可以进一步选择固定大小或可变大小的块来进行重复数据删除。固定大小的分块将数据分区为等大小（逻辑）块，而可变大小的分块（也称为内容定义的分块）通常以内容相关的方式指定块边界（例如，通过 Rabin 指纹 [54]）和分区将数据文件转换为可变大小的块。固定大小的分块简单而快速，而可变大小的分块在内容转换时保持存储效率，因为即使文件添加或删除了一些内容，大多数块仍然保持不变。可变大小的分块通常用在备份系统中（例如，[46]，[66]），但固定大小的分块显示对某些工作负载（例如，VM 映像 [38]）有效。我们的工作涉及固定大小和可变大小的分块方法。加密的重复数据删除解决了外包环境（例如，云存储）中的块机密性，同时保持了重复数据删除的有效性。传统的对称加密要求多个客户端通过其（不同的）密钥加密其文件数据，从而将相同的块转换为不再能够进行重复数据删除的不同加密块。消息锁定加密（MLE）[21] 规范了加密重复数据删除。基线 MLE 实例化基于每个明文的内容（例如，基于块的重复数据删除中的逻辑组块）导出对称密钥（称为 MLE 密钥），并使用 MLE 密钥加密明文以形成密文（例如，加密的逻辑块）。因此，MLE 确保将相同的明文加密为相同的密文。最后，存储系统从每个密文中导出标签并执行重复数据删除。

MLE 可以实现为不同的实例化（参见第 VIII 节）。最受欢迎的是融合加密（CE）[29]，它已在各种存储系统中实现（例如，[3]，[5] - [7]，[9]，[13]，[15]]，[28]，[58]）。CE 的主要思想是从每个明文内容的加密散列中导出 MLE 密钥。注意，一些 MLE 实例 [21] 允许将相同的明文加密成不同的密文。然而，他们确保标签仍然来自明文，因此他们可以通过检查标签的相等性来执行重复数据删除。现有的 MLE 实例都建立在概念确定性加密的基础上，以确保密文（或标签）确定性

地从明文传递，从而保持重复数据删除的有效性，而不是传统的对称加密。虽然确定性加密提供了机密性保证，但在本文中，我们认为攻击者可以利用这种确定性来推断给定密文的原始明文。

第二章 Introduction

Modern storage systems save storage space by removing content redundancy via *deduplication*, which stores only the data copies with unique content among all already stored data copies. Field studies show that deduplication effectively saves substantial fractions of storage space for real-world storage workloads, such as backups [1–4], file system snapshots [5], and virtual machine (VM) disk images [6]. Cloud storage services (e.g., Dropbox and Google Drive) also deploy deduplication to save maintenance costs [7]. To ensure data confidentiality, clients should first encrypt their own data before writing the data to a storage system. It is critical that the encryption approach preserves the original content redundancy pattern, so that duplicate data copies can still be detected and removed from the storage system even after encryption.

We study *encrypted deduplication*, which combines encryption and deduplication in a way that each plaintext (data copy) is symmetrically encrypted and decrypted by a key that is derived from the content of the plaintext itself; in other words, should the plaintexts be duplicates, their resulting ciphertexts are also duplicates and hence can be deduplicated. Encrypted deduplication can be achieved through the formal cryptographic primitive known as *message-locked encryption (MLE)* [8], whose instantiations have been extensively studied and realized in the literature (see Section 第三章). MLE builds on the notion of *deterministic encryption*, which deterministically maps a plaintext to the same ciphertext to preserve the identical content pattern. In addition to MLE, deterministic encryption is also found in symmetric searchable encryption (SSE) [9] and order-revealing encryption (ORE) [10], so as to allow efficient queries over encrypted data.

However, because of the deterministic nature, encrypted deduplication leaks information about the ciphertexts, such that the adversaries can analyze the ciphertexts and infer their corresponding original plaintexts. Specifically, an adversary can perform *frequency analysis* [11] on both the ciphertexts that are observed and the plaintexts that are known a priori. Then the adversary can infer the ciphertext-plaintext pairs, in which both the ciphertext and the plaintext in each pair have correlated frequency patterns. Such inference attacks have been shown effective against database records (e.g., [**?**, 12–16]) and searchable keywords (e.g., [17–21]) that are protected by deterministic encryption. In particular, frequency analysis is shown to effectively infer plaintext information from

encrypted deduplication [22].

In this paper, we revisit the information leakage issues in encrypted deduplication. We argue that encrypted deduplication can actually be even more vulnerable towards carefully crafted frequency analysis attacks. Such sophisticated attacks not only infer a significant fraction of ciphertext-plaintext pairs as shown in the previous work [22], but also ensure that each inferred plaintext indeed corresponds to the target ciphertext with a high probability. From a statistical perspective, The latter implies that the number of false positives is low, or equivalently, the precision is high.

To this end, we propose *two* new frequency analysis attacks against practical encrypted deduplication storage systems, and demonstrate their severity using real-world storage workloads. Both attacks build on the characteristics of practical deduplication workloads. While such characteristics have been exploited to improve deduplication performance and storage efficiency in practice, we show how to exploit them from an adversarial perspective. We hope that our findings help practitioners and researchers better understand the vulnerabilities in encrypted deduplication and motivate more research along this direction.

2.1 Overview of New Frequency Analysis Attacks

We summarize the main ideas of our two new frequency analysis attacks against encrypted deduplication, namely the *distributed-based attack* and the *clustering-based attack*. They build on different assumptions of adversarial knowledge.

2.1.0.0.1 Distribution-based attack: The attack builds on the *locality* property [1, 2, 23], which states that the ordering of data is likely preserved across various backup files, to infer the ciphertext-plaintext pairs that are in similar positions of backup files. In particular, it examines the relative frequency distribution based on the co-occurrence frequencies of adjacent data, and explores the observation that a plaintext and its corresponding ciphertext are likely to have similar relative frequency distributions. Our evaluation shows that the distribution-based attack reduces the number of false positives of the previously proposed locality-based attack [22] by up to 82.4%, while correctly inferring up to 21.2% ciphertext-plaintext pairs (higher than that of the locality-based attack [22] by 6% in the same case).

Furthermore, the distribution-based attack can exploit the leakage of the varying sizes

of plaintexts to filter more false positives. Our evaluation shows that it can suppress the fraction of false positives to as low as 1.7%.

2.1.0.0.2 Clustering-based attack: The attack relaxes the knowledge about the ordering pattern exploited by the distribution-based attack. It builds on the *similarity* property [23, 24], which states that backup files share a large fraction of the same data (the ordering may not be preserved), to infer the ciphertext-plaintext pairs that are clustered in similar regions of a backup file. Our evaluation shows that the clustering-based attack infers up to 30.8% of raw data content in a real-world VM image dataset.

2.1.0.0.3 Observations: We study the nature of our frequency analysis attacks based on real-world datasets. In general, the distribution-based attack preserves the attack severity even when the prior knowledge of plaintexts is loosely correlated with the ciphertexts to be attacked. The reason is that it uses a statistical approach to quantify the relative frequency distribution and addresses the disturbance to frequency ranking. This indicates a wide exploitation of the distribution-based attack, which should be taken into account when developing countermeasures. Also, the distribution-based attack is more likely to affect rarely updated large files (e.g., disk image files) or some correlated small files (e.g., programming source files) that are stored together, while the clustering-based attack mainly threatens the confidentiality of VM disk images.

2.2 Summary of Contributions

We summarize the main contributions of the paper.

- We present two novel frequency analysis attacks against encrypted deduplication. In addition to exploiting the leakage of frequencies due to the deterministic nature of encrypted deduplication, both attacks exploit the characteristics of deduplication workloads to increase the severity of frequency analysis.
- We evaluate our frequency analysis attacks with several real-world datasets, including long-term backups [**?**, 25], Windows file system snapshots [5], and VM disk images [26, 27]. We present new observations based on our evaluation, and further analyze the security implications of how our frequency analysis attacks bring actual damages.
- We discuss possible countermeasures to mitigate the information leakage in encrypted deduplication.

The remainder of the paper proceeds as follows. Section 第三章 presents the basics of encrypted deduplication. Section 第四章 defines the threat model. Sections 第五章 and 第六章 present two new frequency analysis attacks against encrypted deduplication. Section 第七章 presents our trace-driven evaluation results. Section 第八章 discusses the countermeasures against information leakage. Section 第九章 reviews related work, and finally, Section 第十章 concludes the paper.

第三章 Background

In this section, we present the basics of both plain and encrypted deduplication approaches.

We focus on *chunk-based deduplication*, which operates at the granularity of small-size data units called *chunks*. Figure 3-1 summarizes the deduplication workflow. Specifically, a storage system first partitions a file (e.g., backup) of a client into *logical chunks* via some *chunking* procedure, such that each logical chunk is uniquely identified by the cryptographic hash of its content called a *tag* (a.k.a. fingerprint). Two logical chunks are said to be *identical* if they have the same tag, while the likelihood that distinct logical chunks have the same tag is practically negligible [28]. The storage system stores only a copy of identical logical chunks as a *physical chunk*, and each identical logical chunk refers to the same physical chunk via small-size references.

Chunk-based deduplication is more fine-grained than whole-file deduplication, and hence has higher storage efficiency in general. We can further choose either *fixed-size* or *variable-size* chunking for deduplication. Fixed-size chunking partitions file data into equal-size (logical) chunks, while variable-size chunking (a.k.a. content-defined chunking) often specifies chunk boundaries in a content-dependent manner (e.g., via Rabin fingerprinting [**?**]) and partitions file data into variable-size chunks. Fixed-size chunking is simple and fast, while variable-size chunking maintains storage efficiency in the face of content shifts since the majority of chunks remain unchanged even the file has added or removed some content. Variable-size chunking is often used in backup systems (e.g., [1,2]), yet fixed-size chunking is shown to be effective for some workloads (e.g., VM images [6]). Our work addresses both fixed-size and variable-size chunking approaches.

Encrypted deduplication addresses chunk confidentiality in an outsourcing environment (e.g., cloud storage), while preserving deduplication effectiveness. Traditional symmetric encryption requires that multiple clients encrypt their file data by their (distinct) secret keys, thereby transforming identical chunks into distinct encrypted chunks that can no longer be deduplicated. *Message-locked encryption (MLE)* [8] formalizes encrypted deduplication. The baseline MLE instantiation derives a symmetric key (called the *MLE key*) based on the content of each *plaintext* (e.g., a logical chunk in chunk-based deduplication), and encrypts the plaintext using the MLE key to form a *ciphertext* (e.g., an

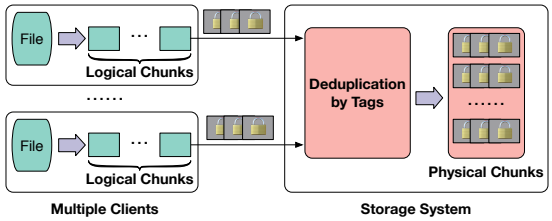


图 3-1 Overview of deduplication workflow.

encrypted logical chunk). Thus, MLE ensures that identical plaintexts are encrypted to identical ciphertexts. Finally, the storage system derives the tag from each ciphertext and performs deduplication.

MLE can be realized as different instantiations (see Section 第九章). The most popular one is convergent encryption (CE) [29], which has been implemented in a wide variety of storage systems (e.g., [**?**, **?**, **?**, **?**, **?**, 30–33]). The main idea of CE is to derive the MLE key from the cryptographic hash of the content of each plaintext. Note that some MLE instantiations [8] allow identical plaintexts to be encrypted into distinct ciphertexts. Nevertheless, they ensure that the tags are still derived from the plaintexts, so that they can perform deduplication by checking the equality of tags.

Existing MLE instantiations all build on the notion *deterministic encryption* to ensure that the ciphertexts (or the tags) are deterministically derived from the plaintexts, so as to preserve deduplication effectiveness as opposed to traditional symmetric encryption. While deterministic encryption provides confidentiality guarantees, in this paper, we argue that such a deterministic nature can be exploited by an adversary to infer the original plaintext from a given ciphertext.

第四章 Threat Model

In this section, we formulate the threat model for our proposed frequency analysis attacks against encrypted deduplication.

4.1 Definitions

We first provide the definitions for our threat model. We model a plain file as an ordered list of *n logical plaintexts* (i.e., logical chunks before deduplication), denoted by $\mathbf{M} = \langle \hat{M}^{(1)}, \hat{M}^{(2)}, \ldots, \hat{M}^{(n)} \rangle$. Each logical plaintext $\hat{M}^{(i)}$ (where $1 \leq i \leq n$) is encrypted via MLE into the corresponding ciphertext $\hat{C}^{(i)}$, and all encrypted results of $\mathbf{M}$ form a stream of *n logical ciphertexts* $\mathbf{C} = \langle \hat{C}^{(1)}, \hat{C}^{(2)}, \ldots, \hat{C}^{(n)} \rangle$. Note that the logical ciphertexts in $\mathbf{C}$ are also ordered to reflect the deduplication processing sequence of an encrypted deduplication storage system.

Identical logical plaintexts and ciphertexts may appear at different positions of $\mathbf{M}$ and $\mathbf{C}$, respectively. We denote a unique plaintext as M (uniquely identified by its tag), which is encrypted via MLE to the corresponding unique ciphertext C. Each M (resp. C) corresponds to one or multiple identical copies of $\hat{M}^{(i)}$ (resp. $\hat{C}^{(i)}$).

4.2 Adversarial Goals and Assumptions

We consider an adversary that aims to infer a set of ciphertext-plaintext pairs, denoted by $\{(C, M)\}$, with two specific goals:

- **High inference rate:** A large fraction of correct ciphertext-plaintext pairs are inferred, among all correct ciphertext-plaintext pairs (i.e., high recall or low false negative rates in statistical terms).
- **High inference precision:** A large fraction of ciphertext-plaintext pairs are correct, among all inferred ciphertext-plaintext pairs (i.e., high precision or low false positive rates in statistical terms).

We assume that the adversary is *honest-but-curious*, such that it can passively monitor the stream of ciphertexts $\mathbf{C}$ being written to the storage system and exploit different types of leakage information from $\mathbf{C}$ (see Section 4.3). Given the available leakage information, the adversary aims to infer the original plaintext of each ciphertext in $\mathbf{C}$ via a *ciphertext-only* attack. It is possible that the adversary knows a limited subset of ciphertext-plaintext

pairs to launch a known-plaintext attack, which further strengthens attack severity [22]. We do not consider the known-plaintext attack in this paper.

We assume that the adversary does not have access to any *metadata* that contains the information about how chunks are operated and stored, as we do not apply deduplication to the metadata and hence it can be protected by traditional symmetric encryption. Also, we assume that the adversary does not have *active* capabilities, which can be independently prevented by existing approaches. One example is that a malicious client may claim the ownership of unauthorized files in client-side deduplication [7,34,35]; it can be addressed by proof-of-ownership [34,36,37] or server-side deduplication [7,26]. Another example is that a malicious storage system may modify stored data; it can be detected by remote integrity checking [38,39].

4.3 Leakage Channels

We consider three types of leakage channels in encrypted deduplication storage that enable an adversary to infer information:

- **Frequency:** Due to the deterministic nature of encrypted deduplication, the frequency of each ciphertext in **C** before deduplication (i.e., the number of duplicate copies) can be mapped to that of its corresponding plaintext in **M**.
- **Order:** Some storage systems (e.g., [1,2,23]) apply deduplication to the chunks in the same order as they appear in the original file. Thus, the order of ciphertexts in **C** can be mapped to that of plaintexts in **M**.
- **Size:** Variable-size chunking (see Section 第三章) leads to different chunk sizes of the ciphertexts. If such ciphertexts are not padded (to avoid storage overhead [40]), the size of a ciphertext in **C** can be mapped to that of its corresponding plaintext in **M**.

In addition, the adversary has access to some *auxiliary information* that presents the ground truth about the data characteristics correlated with **M**. Note that the availability of auxiliary information is necessary for any inference attack [13–15, 18–20, 22, 40, 41]. In this work, we consider the auxiliary information as an ordered list of previously known plaintexts (e.g., old user backups or VM disk images), denoted by **A**. Clearly, the attack severity depends on the correlation between **A** (i.e., the previously known plaintexts) and **M** (i.e., the plaintexts that are to be inferred). Our focus is not to address how an adversary can obtain auxiliary information, possibly due to careless data release [**?**], stolen storage devices [**?**], and cloud leakage [**?**]. Instead, given such information, we investigate how the

available auxiliary information, combined with the leakage channels, bring information leakage from encrypted deduplication.

We focus on frequency analysis [11] as the attack methodology. Classical frequency analysis sorts the unique ciphertexts in **C** and the unique plaintexts in **A** by *frequency* (i.e., the number of identical copies corresponding to each unique ciphertext or unique plaintext). It then relates each unique ciphertext in **C** with the unique plaintext in **A** that has the same frequency rank. In the following sections, we design sophisticated frequency analysis attacks against encrypted deduplication.

第五章 Distribution-based Attack

The distribution-based attack exploits the order information of **C** and **A** to strengthen the effectiveness of frequency analysis. It then compares the relative frequency distributions of **C** and **A** to reduce the false positive results. We also show how to exploit chunk size information to further improve the inference precision.

5.1 Background: Locality-based Attack

The distribution-based attack builds on the previously proposed *locality-based attack* [22], which shows how frequency analysis causes information leakage in backup workloads. The locality-based attack considers *full backups* (or backups for short) that are created as the complete copies of primary data (e.g., application states, file system snapshots, and VM disk images) on a daily or weekly basis. It aims to infer the ciphertext-plaintext pairs across different versions of backups. It assumes that the auxiliary information **A** is derived from some older backups, and its goal is to infer the plaintexts of the latest backup (i.e., **M**).

The locality-based attack leverages the *locality* property [1, 2, 23], which is a common phenomenon in practical backup workloads. Specifically, the locality property states that neighboring chunks tend to co-occur in the *same order* across different versions of backups before deduplication. The main reason is that the updates to each backup are often clustered in some small regions of chunks, while the remaining large stretch of chunks appear unchanged or preserve the same order across different versions of backups.

Based on locality, the locality-based attack exploits the order information to discover the neighboring information of ciphertexts and plaintexts. Specifically, for a given unique ciphertext C, the attack first identifies the set of all identical copies $\{\hat{C}^{(i)}\}$. For each $\hat{C}^{(i)}$, it considers the left and right *neighbors* of $\hat{C}^{(i)}$, i.e., $\hat{C}^{(i-1)}$ and $\hat{C}^{(i+1)}$, respectively. It extracts the sets of left and right neighbors into the associative arrays $\mathbf{L_C}$ and $\mathbf{R_C}$, respectively. The associative arrays store the mappings of each unique ciphertext C and its *co-occurrence frequencies* with its left and right neighbors, respectively. Similarly, the attack also constructs the associative arrays $\mathbf{L_A}$ and $\mathbf{R_A}$ based on the order information of **A**.

The locality-based attack then iterates frequency analysis through the neighbors of each inferred ciphertext-plaintext pair. It first applies frequency analysis to infer a number

(parameterized by u) of top-frequent ciphertext-plaintext pairs $\{(C, M)\}$ from **C** and **A**. The inferred results are likely to be *real* (i.e., the target ciphertext is indeed mapped from the inferred plaintext), based on the observation that the ranks of highly frequent chunks are stable across different versions of backups. For each inferred pair (C, M), the attack finds their left and right neighbors that have the most co-occurrence frequencies with C and M, respectively. Due to locality, the left and right neighbors of M are likely to be the original plaintexts of the corresponding left and right neighbors of C, respectively. Thus, the attack also includes the top-frequent left (resp. right) neighbors of C and M into the set of the resulting inferred ciphertext-plaintext pairs. Finally, the attack procedure iterates until the neighbors of each inferred ciphertext-plaintext pair are examined.

However, the locality-based attack has a major weakness that it introduces a high number of false positives (i.e., incorrect ciphertext-plaintext pairs). Since the main idea of frequency analysis is to map ciphertexts to plaintexts with the same frequency ranks, any disturbance to frequency ranking (e.g., the updates across backups) can lead to incorrect ciphertext-plaintext pairs, which can in turn comprise the inference of their neighbors. Although the locality-based attack is shown to effectively infer a significant fraction of real ciphertext-plaintext pairs, the adversary has low confidence to tell whether each inferred ciphertext-plaintext pair is real or a false positive.

5.2 Description of Distribution-based Attack

The distribution-based attack extends the locality-based attack [22] to significantly remove false positives. It leverages the locality property in backup workloads as in the locality-based attack. In addition, for each unique ciphertext C in **C**, we measure its *relative frequency distribution* based on its co-occurrence frequencies with its neighbors. Similarly, we examine the relative frequency distribution of each unique plaintext M in **A**. Our observation is that for a real ciphertext-plaintext pair (C, M), both C and M should have similar relative frequency distributions; i.e., their co-occurrence frequencies with their respective neighbors are similar. We treat this observation as a more general notion of the locality property, and use it as a condition to filter possibly incorrect ciphertext-plaintext pairs.

Figure 5-1 presents the workflow of the distribution-based attack. We first sort the unique ciphertexts and plaintexts by their frequencies in **C** and **A**, respectively. As in the locality-based attack [22], we configure the parameter u and the underlying frequency

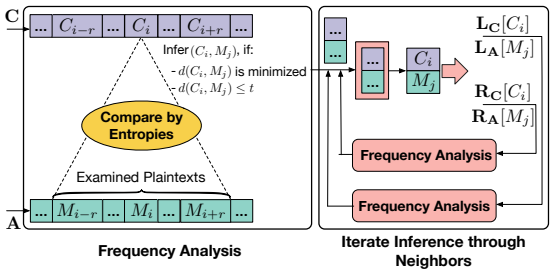


图 5-1 Workflow of distribution-based attack.

analysis to return at most u ciphertext-plaintext pairs. In particular, for each unique ciphertext C_i of rank i (where $1 \leq i \leq u$), we examine a number of unique plaintexts $M_{i-r}, \ldots, M_i, \ldots, M_{i+r}$ that rank from $i-r$ to $i+r$, where r is a configurable parameter (e.g., 10 by default) that indicates the maximum range of rank disturbance that can be addressed.

For each C_i (where $1 \leq i \leq u$) and the corresponding M_j (where $i-r \leq j \leq i+r$), we compare their relative frequency distributions by *entropy*, a key concept in information theory that measures the amount of information produced by a random source. Here, we adopt the entropy to *quantify the randomness of probability distribution* [42]. Specifically, we define two random variables, denoted by X and Y, to describe the co-occurrences of C_i with its left and right neighbors, respectively, such that the event "$X = C$" denotes the case that C is the left neighbor of C_i, and the event "$Y = C$" denotes the case that C is the right neighbor of C_i. Thus, we compute the probabilities of both events based on $\mathbf{L_C}$ and $\mathbf{R_C}$:

$$\Pr[X = C] = \frac{\mathbf{L_C}[C_i][C]}{\sum_{C' \in \mathbf{L_C}[C_i]} \mathbf{L_C}[C_i][C']},$$
$$\Pr[Y = C] = \frac{\mathbf{R_C}[C_i][C]}{\sum_{C' \in \mathbf{R_C}[C_i]} \mathbf{R_C}[C_i][C']},$$

where $\mathbf{L_C}[C_i]$ and $\mathbf{R_C}[C_i]$ store the left and right neighbors of C_i, respectively, while $\mathbf{L_C}[C_i][C']$ and $\mathbf{R_C}[C_i][C']$ store the co-occurrence frequencies of C_i with its left and right neighbor C'. Both X and Y follow the relative frequency distributions of C_i, and we characterize their randomness by entropies denoted by $e(\mathbf{L_C}[C_i])$ and $e(\mathbf{R_C}[C_i])$, respectively:

$$e(\mathbf{L_C}[C_i]) = \sum_{C \in \mathbf{L_C}[C_i]} \log_2 \frac{1}{\Pr[X = C]},$$
$$e(\mathbf{R_C}[C_i]) = \sum_{C \in \mathbf{R_C}[C_i]} \log_2 \frac{1}{\Pr[Y = C]}.$$

Similarly, we compute the entropies $e(\mathbf{L_A}[M_j])$ and $e(\mathbf{R_A}[M_j])$ of each M_j based on $\mathbf{L_A}$ and $\mathbf{R_A}$, respectively. We then quantify the similarity of the relative frequency distributions of C_i and M_j via the *Euclidean distance*, denoted by $d(C_i, M_j)$:

$$\begin{aligned} d(C_i, M_j) &= \Big\{[e(\mathbf{L_C}[C_i]) - e(\mathbf{L_A}[M_j])]^2 \\ &\quad + [e(\mathbf{R_C}[C_i]) - e(\mathbf{R_A}[M_j])]^2\Big\}^{1/2}. \end{aligned}$$

Clearly, C_i and M_j have similar relative frequency distributions only when the Euclidean distance $d(C_i, M_j)$ of their entropies is small. Thus, we identify (C_i, M_j) as an inferred ciphertext-plaintext pair if they satisfy the following requirements:

- **R1:** $d(C_i, M_j)$ is the *smallest* for all $i - r \leq j \leq i + r$.
- **R2:** $d(C_i, M_j)$ is not larger than a pre-defined parameter t (e.g., 1 by default).

One special note is that the original plaintext of C_i may fall outside of the examined plaintexts $M_{i-r}, \ldots, M_{i+r}$. In this case, R1 is still satisfied by some M_j ($i - r \leq j \leq i + r$), and we expect to filter the incorrect ciphertext-plaintext pair by R2.

Then, we adopt the iteration paradigm in the prior locality-based attack [22] (see Section 5.1) to increase the coverage of inferred ciphertext-plaintext pairs. Specifically, for each inferred (C_i, M_j), we apply the new frequency analysis scheme (see above) to infer more ciphertext-plaintext pairs through the neighbors of C_i and M_j, and further iterate the same inference for those newly inferred pairs. We finally stop the iteration, when none of new ciphertext-plaintext pairs can be inferred.

5.2.0.0.1 Summary: To summarize, the distribution-based attack provides a more general notion of locality by considering the relative frequency distribution during frequency analysis. It is configured by three parameters (i) u, which specifies the maximum number of ciphertext-plaintext pairs returned by frequency analysis, (ii) r, which specifies the maximum range of rank disturbance to be considered, and (iii) t, which specifies the Euclidean distance threshold to filter possibly incorrect inference results. The prior locality-based attack [22] can be regarded as a special case of the distribution-based attack under the parameter configuration of $r = 0$ (i.e., without addressing any disturbance to frequency ranking) and $t \to \infty$ (i.e., without filtering any incorrect inference results).

5.3 Exploiting Size Leakage

We propose an advanced variant of the distribution-based attack that operates with size information to further reduce false positives. Specifically, we assume that the size of each ciphertext in $\mathbf{C}$ reflects that of its original plaintext.

Our idea builds on the fundamental truth that if a ciphertext C corresponds to a plaintext M, then the size of C approximates that of M. This is because encrypted deduplication preserves the number of blocks (i.e., the basic units operated by symmetric encryption) in the content to be encrypted. We use this fact to further filter the incorrect ciphertext-plaintext pairs (C, M), where the number of blocks in C and M are different.

In this work, we assume that each block is of 16 bytes, which is a typical configuration for AES. For each considered ciphertext-plaintext pair (C_i, M_j), we derive the number of blocks in C_i and M_j as $b(C_i)$ and $b(M_j)$, respectively:

$$b(C_i) = \lceil \frac{\mathsf{size}(C_i)}{16} \rceil,$$
$$b(M_j) = \lceil \frac{\mathsf{size}(M_j)}{16} \rceil,$$

where $\mathsf{size}(C_i)$ and $\mathsf{size}(M_j)$ are the exact sizes of C_i and M_j, respecitively, and $\lceil x \rceil$ returns the smallest integer greater than or equal to x. In addition to R1 and R2 (see Section 5.2), we apply the following requirement to filter by size:

- **R3:** $b(C_i)$ equals $b(M_j)$.

The requirement R3 is effective to filter the incorrect ciphertext-plaintext pairs regarding variable-size chunks, where different chunks possibly have distinct sizes. However, for fixed-size chunks, it is always satisfied by the ciphertext-plaintext pairs, even they are incorret. Despite of this, we can still apply R1 and R2 to achieve high-precision attack.

第六章 Clustering-based Attack

In this section, we relax the adversarial knowledge in the distribution-based attack, and propose the *clustering-based attack*, which does not need the fine-grained ordering of plaintexts. Instead, it exploits the *similarity* property to infer original data from similar *clusters* (i.e., some large data units aggregated by chunks), without relying on the ordering of chunks in each cluster. We first introduce similarity, and then show how to adapt this property into inference attack.

6.1 Background: Similarity

Similarity [24] states that backup files from the same source are likely to be *similar* and share a large fraction of identical chunks. The similarity between backup files can be quantified by *Broder's theorem* [43]. Specifically, if we treat each file as a set S of chunks (i.e., ignore their orders), Broder's theorem states that if the probability that two sets of chunks share the same minimum chunk hash is high, then both sets are likely to share a large fraction of identical chunks and vice versa:

$$\Pr[\min\{\mathrm{H}(S)\} = \min\{\mathrm{H}(S')\}] = \frac{|S \cap S'|}{|S \cup S'|},$$

where $\mathrm{H}(\cdot)$ is a hash function that is chosen uniformly at random from a min-wise independent family of permutations, and $\min\{\mathrm{H}(S)\}$ returns the minimum chunk hash of S.

Prior works that target different aspects (e.g., performance [23, 24, 27], and security [22]) of deduplication have applied similarity to *preserve storage efficiency*. Specifically, they operate deduplication just on the files that share the same minimum chunk hash (i.e., similar); since similar files can have a large number of identical chunks due to Broder's theorem, such *near-exact* deduplication only leads to a slight degradation of storage efficiency. Different from prior approaches [22–24, 27], we apply similarity to increase the severity of frequency analysis.

6.2 Description of Clustering-based Attack

We now present the clustering-based attack (Figure 6-1) that builds on similarity to infer original data against encrypted deduplication.

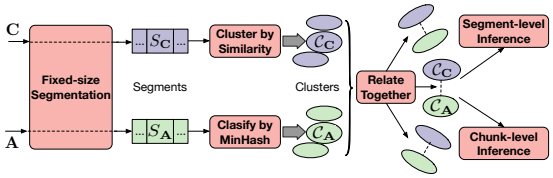


图 6-1 Workflow of clustering-based attack: MinHash denotes the minimum chunk hash of a plaintext segment.

To exploit similarity, we first introduce *segmentation* on a stream of chunks. Specifically, we partition **C** into a set of coarse-grained data units, called *ciphertext segments*. Each ciphertext segment, denoted by $S_{\mathbf{C}}$, comprises multiple adjacent ciphertexts in **C**. In this work, we implement the *fixed-size* segmentation scheme to ensure that all ciphertext segments are of the same size (e.g., 4MB by default). For compatibility, the same fixed-size segmentation scheme also applies to the auxiliary information **A**, so as to generate multiple *plaintext segments*, each of which is denoted by $S_{\mathbf{A}}$. Note that some variable-size segmentation schemes [2, 27] identify segment boundaries after the chunks whose contents match a specific pattern and thus address the boundary shift problem faced by the fixed-size segmentation scheme. However, we cannot use these variable-size segmentation schemes [2, 27] in the attack. The reason is that the original contents of chunks in ciphertext segments are protected by symmetric encryption, and we cannot ensure that the boundaries for both ciphertext and plaintext segments match the same pattern. This leads to an incompatibility between ciphertext and plaintext segments, and degrades the amount of segment-level data inferred by the clustering-based attack (see the late part of this section).

We propose to infer ciphertext-plaintext pairs through similar segments. Let $S_{\mathbf{M}}$ be the original plaintext segment of a ciphertext segment $S_{\mathbf{C}}$ (i.e., each plaintext in $S_{\mathbf{M}}$ corresponds to some ciphertext in $S_{\mathbf{C}}$ and vice versa). According to Broder's theorem, if $S_{\mathbf{M}}$ shares the same minimum chunk hash, say h, with some plaintext segment $S_{\mathbf{A}}$, then $S_{\mathbf{M}}$ and $S_{\mathbf{A}}$ tend to have a large fraction of identical plaintexts. This implies that the ciphertexts in $S_{\mathbf{C}}$ are likely to be mapped from the plaintexts in $S_{\mathbf{A}}$. In other words, we first classify all plaintext segments of **A** by their minimum chunk hashes, and obtain multiple *plaintext clusters*. Each plaintext cluster, denoted by $\mathcal{C}_{\mathbf{A}} = \{S_{\mathbf{A}}\}$, corresponds to a unique minimum chunk hash shared by its included segments. We also group the ciphertext segments, whose original plaintext segments have the same minimum chunk hash h (i.e., similar), into an identical *ciphertext cluster*, denoted by $\mathcal{C}_{\mathbf{C}} = \{S_{\mathbf{C}}\}$. Then, we infer the original

data of $\mathcal{C}_{\mathbf{C}}$ from some $\mathcal{C}_{\mathbf{A}}$ that corresponds to h.

We propose a *clustering* scheme to group similar ciphertext segments. A naïve approach is to classify segments by their minimum chunk hashes, but it does not work on ciphertext segments, whose original contents are protected by symmetric encryption. We address the classification issue based on Broder's theorem. Our insight is that if two ciphertext segments have a large fraction of identical ciphertexts, then they correspond to the same minimum chunk hash with a high probability. The reason is that deterministic encryption preserves the cardinalities of union and intersection of plaintext segments, based on which we can use Broder's theorem to learn the equivalence of their minimum chunk hashes.

In this work, we define the *clustering distance* $d(S_{\mathbf{C}}, S'_{\mathbf{C}})$ of any two ciphertext segments $S_{\mathbf{C}}$ and $S'_{\mathbf{C}}$ by one minus the fraction of their identical ciphertexts:

$$d(S_{\mathbf{C}}, S'_{\mathbf{C}}) = 1 - \frac{|S_{\mathbf{C}} \cap S'_{\mathbf{C}}|}{|S_{\mathbf{C}} \cup S'_{\mathbf{C}}|}.$$

Note that identical ciphertexts may repeat in $S_{\mathbf{C}}$ or $S'_{\mathbf{C}}$, and $|S_{\mathbf{C}} \cap S'_{\mathbf{C}}|$ and $|S_{\mathbf{C}} \cup S'_{\mathbf{C}}|$ return the number of *unique* ciphertexts in their intersection and union, respectively. Clearly, the smaller $d(S_{\mathbf{C}}, S'_{\mathbf{C}})$ is, the more likely are $S_{\mathbf{C}}$ and $S'_{\mathbf{C}}$ to correspond to the same minimum chunk hash. Then, we adopt the *agglomerative hierarchical clustering (AHC)* [44] to aggregate similar ciphertext segments based on their distance information. Specifically, we start with eash ciphertext segment in its own singleton cluster, and iteratively combine the two closest clusters based on the maximum distance of their ciphertext segments. We configure a parameter k, and stop the iterated combination when the maximum distance of ciphertext segments in the two cloest clusters is greater than k.

For each aggregated ciphertext cluster $\mathcal{C}_{\mathbf{C}}$, we relate it to some plaintext cluster $\mathcal{C}_{\mathbf{A}}$ with frequency analysis, while taking frequency distribution into account. This is based on the observation that identical ciphertexts (resp. plaintexts) may repeat in the same or different ciphertext (resp. plaintext) segments and identical ciphertext (resp. plaintext) segments may also repeat in the same ciphertext (resp. plaintext) cluster. We propose to examine the frequency distribution of the logical ciphertexts or plaintexts in each cluster, and perceive that the frequency distributions for similar clusters (i.e., correspond to the same minimum chunk hash) are also likely to be similar.

We proceed the frequency analysis scheme as follows. First, we sort available ciphertext and plaintext clusters by the total number of logical ciphertexts and plaintexts they

include, respectively. Then, we count an associative array **F** that stores the frequency of each unique ciphertext or plaintext in corresponding cluster. Based on **F**, we compute the probability that a ciphertext C exists in a ciphertext cluster $\mathcal{C}_{\mathbf{C}}$ and further the entropy of $\mathcal{C}_{\mathbf{C}}$:

$$\Pr[C \in \mathcal{C}_{\mathbf{C}}] = \frac{\mathbf{F}[\mathcal{C}_{\mathbf{C}}][C]}{\sum_{C' \in \mathcal{C}_{\mathbf{C}}} \mathbf{F}[\mathcal{C}_{\mathbf{C}}][C']},$$
$$e(\mathcal{C}_{\mathbf{C}}) = \sum_{C \in \mathcal{C}_{\mathbf{C}}} \log_2 \frac{1}{\Pr[C \in \mathcal{C}_{\mathbf{C}}]},$$

where $\mathbf{F}[\mathcal{C}_{\mathbf{C}}][C]$ stores the frequency of C in $\mathcal{C}_{\mathbf{C}}$. Similarly, we compute the entropy $e(\mathcal{C}_{\mathbf{A}})$ for the plaintext cluster $\mathcal{C}_{\mathbf{A}}$. Like the distribution-based attack (see Section 5.2), we configure the frequency analysis scheme with the parameters (u, r, t), and infer that the ciphertext cluster $\mathcal{C}_{\mathbf{C}}$ is similar to a plaintext cluster $\mathcal{C}_{\mathbf{A}}$, if they satisfy the following requirements:

- The rank of $\mathcal{C}_{\mathbf{C}}$ is not larger than u.
- The rank difference of $\mathcal{C}_{\mathbf{C}}$ and $\mathcal{C}_{\mathbf{A}}$ is not larger than r.
- The difference of $e(\mathcal{C}_{\mathbf{C}})$ and $e(\mathcal{C}_{\mathbf{A}})$ is the smallest and also not larger than t.

Then, for each pair $(\mathcal{C}_{\mathbf{C}}, \mathcal{C}_{\mathbf{A}})$ of similar clusters, we infer ciphertext-plaintext pairs in two levels.

- **Segment-level inference:** If $\mathcal{C}_{\mathbf{C}}$ and $\mathcal{C}_{\mathbf{A}}$ have the same number of logical chunks (i.e., ciphertexts or plaintexts), as well as an identical entropy, this implies that $\mathcal{C}_{\mathbf{C}}$ is exactly mapped from $\mathcal{C}_{\mathbf{A}}$ with a high probability. In this case, we operate attack on the coarse-grained *segment* level. Specifically, we first compute the entropies of each ciphertext segment $S_{\mathbf{C}}$ in $\mathcal{C}_{\mathbf{C}}$ and each plaintext segment $S_{\mathbf{A}}$ in $\mathcal{C}_{\mathbf{A}}$, based on the frequency distributions of their ciphertexts and plaintexts, respectively. We infer that $S_{\mathbf{C}}$ is mapped from $S_{\mathbf{A}}$, if the total numbers of logical chunks, as well as the entropies, of $S_{\mathbf{C}}$ and $S_{\mathbf{A}}$ are identical. Our evaluation shows that the segment-level inference contributes most of the correctly inferred contents in the clustering-based attack (see Section 7.3). It is also possible to further recover each plaintext in these inferred segments with additional adversarial knowledge (e.g., ordering).
- **Chunk-level inference:** If $\mathcal{C}_{\mathbf{C}}$ and $\mathcal{C}_{\mathbf{A}}$ have different numbers of logical chunks or entropies, we apply frequency analysis to operate attack on the fine-grained *chunk* level. Specifically, we sort all unique ciphertexts and plaintexts by their frequencies in $\mathcal{C}_{\mathbf{C}}$ and $\mathcal{C}_{\mathbf{A}}$, respectively, and infer ciphertext-plaintext pairs based on frequency ranks. However, we find the chunk-level inference does not perform well in our experimental

dataset. The possible reason is each cluster includes a large number of logical chunks, which degrades the effectiveness of frequency analysis. Even so, we expect the chunk-level inference can recover more ciphertext-plaintext pairs in practice, especially when the number of logical chunks in some clusters is limited.

6.2.0.0.1 Summary: To summarize, the clustering-based attack exploits similarity, and launches frequency analysis in similar clusters to infer ciphertext-plaintext pairs. In addition to u, r and t, it is configured by the parameter k, which specifies the upper bound distance in combining the closest clusters.

Although possibly affected by the boundary shift of fixed-size segments, we argue that the clustering-based attack is severe against VM disk images. Specifically, a flat VM image file is allocated of a fixed size at the time it is created, and such size cannot be changed during its lifetime. All unused regions in a VM image are initially filled with zero chunks, which can be further re-written for storing additional data in the image. In Section 7.3, we examine the effectiveness of the clustering-based attack against VM images.

表 7-1 Characteristics of experimental datasets.

Dataset	Category	Characteristics in Each Snapshot	
		#Logical (Million)	#Unique (Million)
FSL	User004	1.0-1.3	0.7-0.9
	User007	3.5-5.2	2.3-3.6
	User012	25.0-26.4	8.9-9.7
	User013	1.8-5.7	1.2-4.2
	User015	13.4-20.5	9.0-11.0
	User028	6.0-10.3	3.5-6.8
MS	Win7	61.6-61.8	61.1-61.3
	Serv-03	10.6-10.7	8.4-8.5
	Serv-08	∼6.5	∼3.8
	Vista-B	∼7.6	∼2.0
	Vista-U	∼21.0	∼10.4
VM	User1	2.6	∼0.9
	User2		1.1-1.3
	User3		0.9-1.1
	User4		∼0.9
	User5		0.9-1.0
	User6		0.9-1.1

The notations #Logical and #Unique denote the total number of logical and unique chunks in each experimental snapshot, respectively.

第七章 Experiments

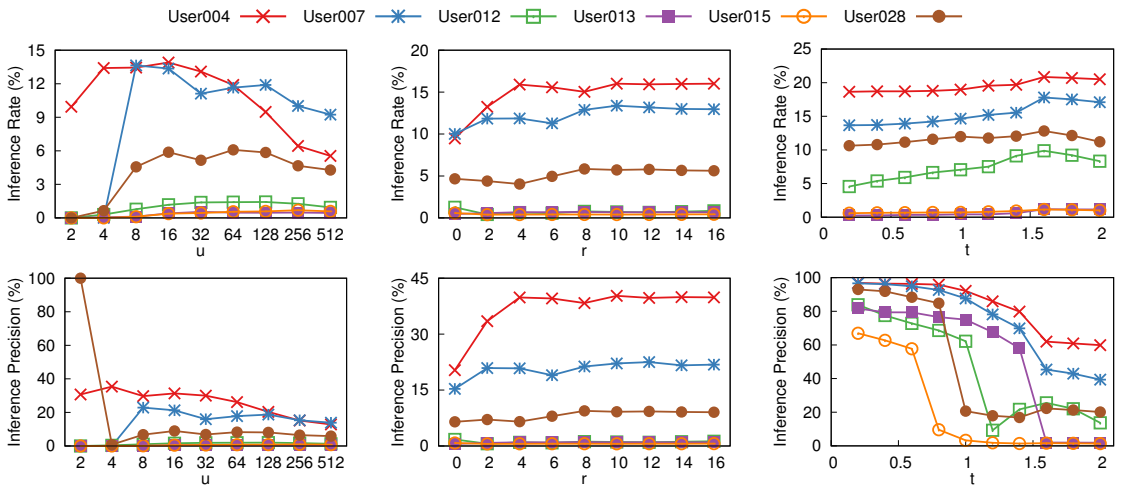
In this section, we present the trace-driven evaluation results to demonstrate the severity of our proposed frequency analysis attacks against encrypted deduplication.

7.1 Methodology

We evaluate our attacks using the following real-world datasets, whose characteristics are summarized in Table 7-1.

- **FSL:** This dataset is collected by the File systems and Storage Lab (FSL) at Stony Brook University [**?**, 25, 45]. We focus on the *fslhomes* snapshots, each of which includes an ordered list of 48-bit chunk hashes that are produced by variable-size chunking under an average size of 8KB, as well as the corresponding metadata information (e.g., chunk size, file name extension, etc). We pick the snapshots from January 22 to May 21 in 2013, and select six users (i.e., User004, User007, User012, User013, User015, and User028) that have the complete daily snapshots over the whole duration. We collect these snapshots on a weekly basis, and hence form 14 weekly full backups for each user.
- **MS:** This dataset is collected by Microsoft [5] and publicized on SNIA [**?**]. The original dataset contains the Windows file system snapshots that span from September 5 to October 31, 2009. Each snapshot is represented by the 40-bit chunk hashes under different average sizes obtained from Rabin fingerprinting [**?**]. We focus on the snapshots that have been installed with the operating systems in the following categories: Windows 7 (Win7), Microsoft Windows Server 2003 (Serv-03), Microsoft Windows Server 2008 (Serv-08), Microsoft Windows Vista Business Edition (Vista-B), and Microsoft Windows Vista Ultimate Edition (Vista-U). In each category, we pick four snapshots, each of which is configured with the average chunk size of 8KB.
- **VM:** This dataset is collected from a university programming course in Spring 2014. The original dataset includes a number of VM image snapshots for the students enrolled in the course, and each snapshot has a fixed size of 10GB and is represented by the ordered list of SHA-1 hashes of 4KB fixed-size chunks. We focus on 6 users (i.e., User1-User6) and extract their snapshots into 13 weekly backups.

Our datasets do not contain actual content, so we mimic the adversarial knowledge based on chunk hashes. Specifically, we use the ordered lists of chunk hashes in some snapshots as the auxiliary information **A** and the ground truth **M**, respectively. To simulate encryption, we apply an additional hash function over each original chunk hash (that represents a plaintext) in **M**, and truncate the result into an agreed number of bits specific to the used dataset. The truncated result mimics a ciphertext in **C**. For each inferred ciphertext-plaintext pair (C, M), we verify its correctness by applying the same simulated encryption on M and comparing the result with C. Note that the clustering-based attack can operate at the segment level, and infer the ciphertext-plaintext segment pair $(S_{\mathbf{C}}, S_{\mathbf{A}})$. In this case, we check $(S_{\mathbf{C}}, S_{\mathbf{A}})$ by examining each ciphertext in $S_{\mathbf{C}}$ is exactly mapped from each plaintext in $S_{\mathbf{A}}$.



(a) Impact of u: $r = 0$; $t \to \infty$

(b) Impact of r: $u = 128$ for User004, User013 and User015, and $u = 256$ for User007, User012 and User028; $t \to \infty$

(c) Impact of t: $u = 128$ for User004, User013 and User015, and $u = 256$ for User007, User012 and User028; $r = 10$

图 7-1 Experiment 1 (Impact of parameters): impact of (u, r, t) in distribution-based attack.

We measure the severity of the attacks by the inference rate and the inference precision (see Section 第四章).

7.2 Results of Distribution-based Attack

7.2.0.0.1 Experiment 1 (Impact of parameters): We evaluate the impact of the parameters (u, r, t) in the distribution-based attack. We drive our evaluation using the FSL dataset, and use the 12th weekly backup of each user as the auxiliary information to infer original plaintexts in corresponding 14th weekly backup. We configure $t \to \infty$ and $r = 0$ to evaluate the impact of u (in this case, the distribution-based attack reduces to the locality-based attack [22]). Our rationale is to avoid the disturbances by other parameters.

Figure 7-1(a) shows the impact of u, when we vary u from 2 to 512. Regarding inference rate, we have the same observation as the prior work [22]. Specifically, the inference rates first increase with u, since the attack can infer more ciphertext-plaintext pairs. After hitting the maximum values (e.g., 13.9% for User004, 13.6% for User007, 1.4% for User012, 0.5% for User013, 0.7% for User015, and 6.1% for User028), they decrease. The reason is that the underlying frequency analysis introduces a large number of false positives that compromise the inferences over corresponding neighbors.

The prior work [22] does not report the inference precision about the attack. We

observe that the inference precisions for all users are at a fairly low level (e.g., less than 40%), except the case of $u = 2$ for User028 that does not introduce any false positives. On the other hand, the inference rate in the special case is about 0.0001%, meaning that the attack only infers a few ciphertext-plaintext pairs. In addition, as u increases, the inference precisions decrease slightly. For example, when u increases from 2 to 512, the inference precision of User004 drops from 30.7% to 12.8%.

Observation (1) – *A relatively larger u increases the inference rate, yet it decreases the inference precision (i.e., more false positives are introduced).*

Informed by the impact of u, we set u at 128 for User004, User013 and User015, and at 256 for User007, User012 and User028, respectively, in order to evaluate the impact of r and t. Our rationale is to increase the coverage of inferred ciphertext-plaintext pairs, while we use r and t to filter possibly false positives.

We first configure $t \to \infty$ and evaluate the impact of r. Figure 7-1(b) shows the results. We observe that the inference rates of majority users increase with r. For example, when we vary r from 0 to 16, the inference rates grow from 9.5% to 16.0%, from 10.0% to 13.0%, from 0.5% to 0.7%, and from 4.7% to 5.6% for User004, User007, User013, and User028, respectively. The reason is that the distribution-based attack addresses disturbances to frequency ranking, and infer more correct ciphertext-plaintext pairs. On the other hand, the inference rates decrease slightly from 1.3% to 0.8% and from 0.6% to 0.4% for User012 and User015, respectively. The reason is that they examine a large range of plaintexts and may introduce more false positives. In addition, the inference precisions for all users are at a low level (e.g., less than 45%), and have similar tendencies with corresponding inference rates.

Observation (2) – *A larger r provides more opportunities of identifying correct ciphertext-plaintext pairs, yet it also increases the probability of having false positives.*

Then, we fix $r = 10$ and evaluate the impact of t. Figure 7-1(c) shows the results. When t is small (e.g., less than 0.5), we observe that the attack misjudges and filters a significant number of ciphertext-plaintext pairs, even they are correct. This introduces *false negatives* that reduce the inference rate. As t increases, the number of false negatives decreases. When $t = 1.5$, the inference rates hit the maximum values at 21.2%, 18.2%, 10.4%, 1.2%, 1.2%, and 13.5% for User004, User007, User012, User013, User015, and User028, respectively. When t further increases to 2, the corresponding inference rates drop to 20.5%, 17.1%, 8.3%, 1.2%, 1.0%, and 11.2%, respectively. The reason is that if t

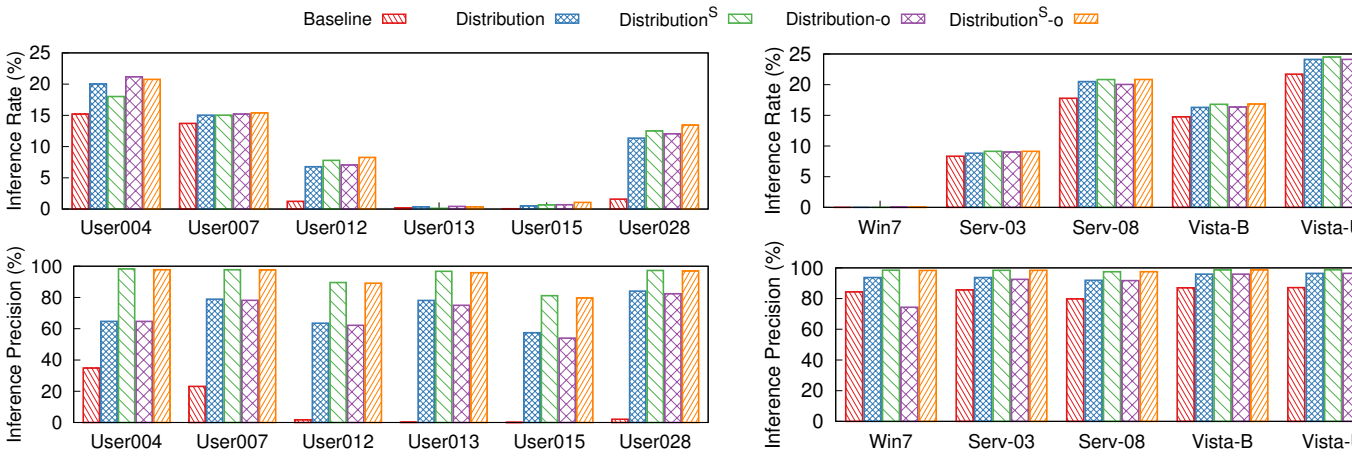
is too large, it cannot filter false positives effectively. For the same reason, the inference precisions for all users decrease with t.

Observation (3) – *A smaller t filters a large fraction of false positives, yet it introduces more false negatives.*

7.2.0.0.2 Experiment 2 (Comparison with prior attack): We compare the severity of the distribution-based attack with that of the locality-based attack [22]. In addition to using the FSL dataset like Experiment 1, we include the MS dataset for cross-dataset validation. Specifically, for each MS category, we choose two snapshots at a time, use one to infer the other, and evaluate the average inference rate and precision.

We consider the following attack instances for comparison.

- `Baseline`: We re-implement the locality-based attack based on the parameter configuration suggested in [22]. Specifically, it infers 5 most frequent ciphertext-plaintext pairs in the first invocation (i.e., to initialize a set of ciphertext-plaintext pairs for iteration) of frequency analysis, and 30 in each following invocation (i.e., to iteratively infer ciphertext-plaintext pairs from neighbors).
- `Distribution` and $\texttt{Distribution}^{\texttt{S}}$: We consider two attack instances of the distribution-based attack, denoted by $\texttt{Distribution}^{\texttt{S}}$ and `Distribution`, which operate with and without size information, respectively (i.e., the superscript `S` indicates that the attack instance operates with size information). We configure $\texttt{Distribution}^{\texttt{S}}$ and `Distribution` under the same configuration of `Baseline`. In addition, we choose r and t in both $\texttt{Distribution}^{\texttt{S}}$ and `Distribution` in the following way: for the FSL dataset, we fix $r = 10$ for all users, and individually set $t = 1.5$, 1.2, 1, 1, 0.7, and 0.9 for User004, User007, User012, User013, User015, and User028, respectively; for the MS dataset, we also fix $r = 10$ for all categories, and set $t = 2$ for Win7 and Serv-08, and $t = 1.6$ for Vista-U, Serv-03, and Vista-B, respectively. This is informed by our tests for optimal configurations of the datasets.
- `Distribution-o` and $\texttt{Distribution}^{\texttt{S}}\texttt{-o}$: We consider two additional distribution-based attack instances, denoted by `Distribution-o` and $\texttt{Distribution}^{\texttt{S}}\texttt{-o}$, which apply the same configurations for r and t as with `Distribution` and $\texttt{Distribution}^{\texttt{S}}$, and further use larger u to increase the coverage of inferred ciphertext-plaintext pairs. Specifically, we configure u in `Distribution-o` and $\texttt{Distribution}^{\texttt{S}}\texttt{-o}$ in the following way: for the FSL dataset, we apply the same configuration of u in Experiment 1; for the MS dataset, we set $u = 128$ for Win7, and $u = 30$ for Serv-03, Serv-08, Vista-B, and Vista-U.



(a) FSL dataset (b) MS dataset

图 7-2 Experiment 2 (Comparison with prior attack): comparison of attack severity for distribution-based attack and locality-based attack.

Figure 7-2(a) shows the comparison results of the FSL dataset. We observe that different instances of the distribution-based attack outperform the locality-based attack in almost all cases. For example, regarding User028, the lowest inference rate of the distribution-based attack is 11.4%, with the precision of 84.1% (due to `Distribution`), while the corresponding inference rate and precision of `Baseline` are only 1.2% and 1.7%, respectively; this implies that the distribution-based attack reduces the number of false postives by 82.4% in this case.

Observation (4) – *The distribution-based attack significantly increases the inference precision, while achieving a higher inference rate than the locality-based attack.*

`Distribution`$^{\mathrm{S}}$ and `Distribution`$^{\mathrm{S}}$`-o` have higher inference precisions than `Distribution` and `Distribution-o`, respectively, since they further filter false positives by size information. For example, for User004, `Distribution`$^{\mathrm{S}}$ and `Distribution`$^{\mathrm{S}}$`-o` reduce the fraction of false positives in `Distribution` and `Distribution-o` from 35.2% to 1.7% and from 35.3% to 2.2%, respectively. However, in the same case, we observe that the inference rates of `Distribution` and `Distribution-o` are 20.0% and 21.2%, slightly higher than those of `Distribution`$^{\mathrm{S}}$ and `Distribution`$^{\mathrm{S}}$`-o` by 2.0% and 0.4%, respectively. The reason is that `Distribution` and `Distribution-o` infer a small number of correct results from the neighbors of incorrect ciphertext-plaintext pairs. In other words, although (C, M) is an incorrect ciphertext-plaintext pair, the neighbors of C may correspond to those of M with a small probability. Even in this case, all distribution-based

attack instances are more severe than the locality-based attack instance. Specifically, the inference rate of `Baseline` is only 15.2%, lower than those of the best and the worst distribution-based attack instances by 6.0% and 2.8%, respectively.

Observation (5) – *Filtering incorrect inference results improves the inference precision, yet it degrades the coverage of inferred ciphertext-plaintext pairs and possibly decreases the inference rate.*

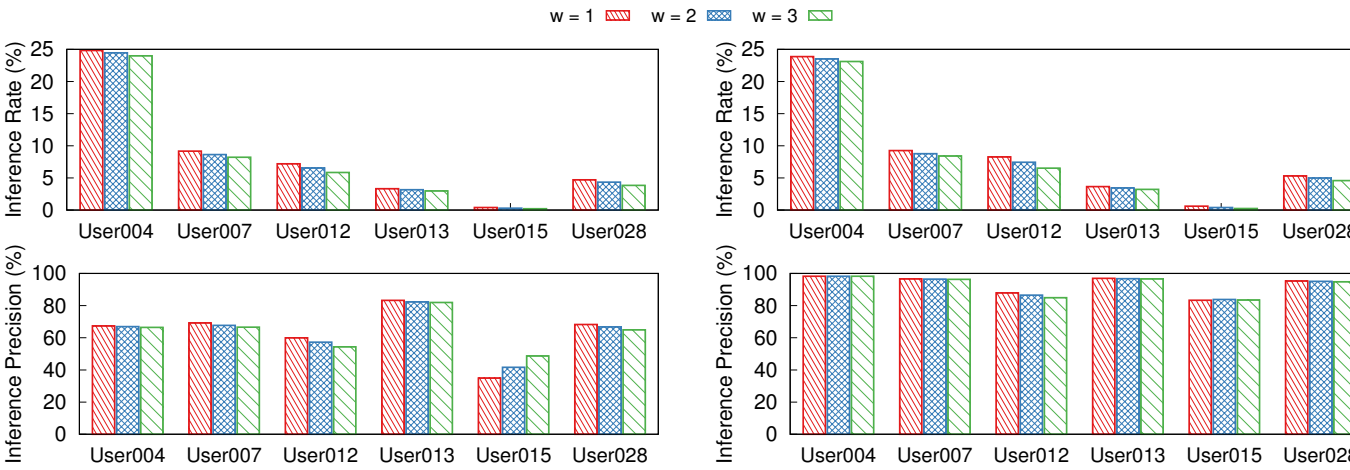
We further observe that although `Distribution-o` and `Distribution`$^{\mathtt{S}}$`-o` build on larger u, their inference rates are just slightly higher than those of `Distribution` and `Distribution`$^{\mathtt{S}}$ by 0.4% and 0.9%, respectively. The reason is that the distribution-based attack iterates inference just through neighbors, and has a bounded coverage of inferred ciphertext-plaintext pairs. The further increase of u only adds a small number of new correct ciphertext-plaintext pairs into results.

Figure 7-2(b) shows the results of the MS dataset. Both locality-based and distribution-based attacks have high inference rates and precisions in most MS categories (except Win7). The possible reason is that MS snapshots are highly correlated (e.g., the variance of the total number of chunks is small, as shown in Table 7-1). We observe that the distribution-based attack still outperforms the locality-based attack. For example, in Vista-U, the inference rates and precisions of all inferences of the distribution-based attack are above 24.1% and 96.4%, while those of `Baseline` are 21.7% and 87.1%, respectively.

Note that both distribution-based and locality-based attacks have low inference rates (e.g., less than 0.01%) in the Win7 category. The reason is that Win7 includes a large fraction (e.g., more than 98.8%, as shown in Table 7-1) of unique chunks, which cannot be correctly inferred by the frequency analysis attacks.

7.2.0.0.3 Experiment 3 (Attack effectiveness): We consider a long-term backup scenario and examine the effectiveness of the distribution-based attack with the FSL dataset. Specifically, we choose the ith FSL weekly backup of each user as the auxiliary information to infer original plaintexts in the corresponding $(i+w)$th FSL weekly backup. Clearly, the smaller w is, the higher correlation between the auxiliary information and the target backup will be. We configure the two distribution-based attack instances `Distribution-o` and `Distribution`$^{\mathtt{S}}$`-o` like Experiment 2, and evaluate their inference rates and inference precisions that are averaged for all available i for each user.

Figure 7-3(a) shows the results. The distribution-based attack has varying inference rate and precision across users. For example, in the favorable case like User004,



(a) Distribution-based attack without size information

(b) Distribution-based attack with size information

图 7-3 Experiment 3 (Attack effectiveness): severity of distribution-based attack in FSL dataset.

it achieves the highest inference rates of 24.8%, 24.5%, and 24.0% with the precisions of 67.3%, 67.0%, and 66.5% for w = 1, 2, and 3, respectively; in the non-favorable case like User015, the inference rate of the distribution-based attack is only around 0.3%. The possible reason is that the backup data from User015 has low chunk locality.

In addition, we observe that the correlation (i.e., w) of the auxiliary information has low impact on the effectiveness of the distribution-based attack. For example, when w increases from 1 to 3, it only leads to limited degradations on inference rate (e.g., less than 1.4%) and precision (e.g., less than 5.6%). The reason is that the distribution-based attack addresses disturbances to frequency ranking and preserves attack effectiveness.

Observation (6) – *The distribution-based attack can limit the degradation of attack effectiveness in the presence of loosely correlated auxiliary information.*

Figure 7-3(b) shows the results of `Distribution`$^{\text{S}}$`-o`. We observe that it has similar inference rate with `Distribution-o`, while achieving much higher precision. For example, the average inference precision of all users are 93.1%, 92.8%, and 92.4% for w = 1, 2, and 3, respectively.

7.3 Results of Clustering-based Attack

7.3.0.0.1 Experiment 4 (Impact of parameter): We first evaluate the impact of the parameter k, which defines the upper bound distance in combining the closest clusters. We use both the FSL and the VM datasets to study how k affects the underlying clustering

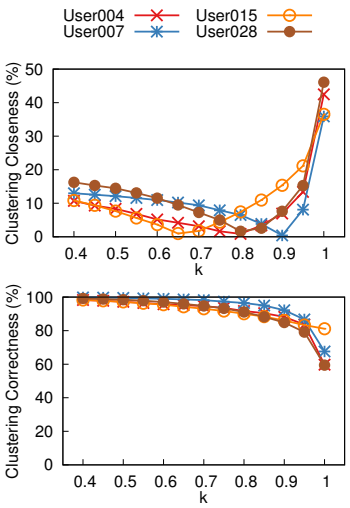
scheme in the attack. Specifically, we apply segmentation on the last backup of each considered FSL and VM user, respectively, and generate the segments that have a fixed size of 4MB.

The clustering scheme aims to aggregate similar ciphertext segments into the same cluster, without compromising the confidentiality of chunks in each ciphertext segment. To quantify its effect, we compare the results of clustering with those of a *real* classification approach, which directly classifies segments by their minimum chunk hash. Suppose we generate m real classes of segments by classification, and $\widetilde{m}$ clusters by the clustering scheme, respectively. We consider *clustering closeness*, evaluated by $\frac{\mathsf{abs}(m-\widetilde{m})}{m}$ (where $\mathsf{abs}(m-\widetilde{m})$ returns the absolute value of $m-\widetilde{m}$), which quantifies how the number of clusters approximates that of real classes. In addition, let $\hat{m}$ be the number of clusters, in which all segments are similar (i.e., have the same minimum chunk hash). We also consider *clustering correctness*, evaluated by $\frac{\hat{m}}{\widetilde{m}}$, which quantifies how precisely the clustering scheme groups similar segments.

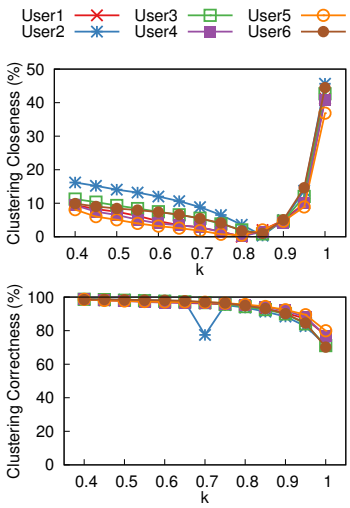
Figure 7(a) shows the results of the FSL dataset, where we consider four FSL users (e.g., User004, User007, User015 and User028) for saving evaluation time. The clustering closeness first increases with k, since the number (i.e., $\widetilde{m}$) of clusters decreases and approximates m. When k increases further, the number of clusters drops away from m, and leads to the increase of clustering closeness. In addition, we observe that the clustering correctness gradually decreases with k, because some of non-similar segments (i.e., their minimum chunk hashes are different) are aggregated into the same cluster. Both results suggest that we can configure an appropriate k to balance the closeness and correctness of clustering. For example, when we set $k = 0.65$ for User015, the corresponding closeness and correctness are 1.0% and 94.2%, respectively. This implies that the results of the clustering scheme highly approximates those of real classification.

Observation (7) – *By configuring an appropriate k for clustering, we approximate the results of classifying segments, without the knowledge of minimum chunk hash in each segment.*

Figure 7(b) shows the results of the VM dataset. We observe that the clustering closeness and correctness of all VM users have similar tendencies with those of FSL users. When we configure $k = 0.8$, the average clustering closeness of all VM users is only 3.0%, while the corresponding clustering correctness is as high as 93.1%.



(a) FSL dataset



(a) VM dataset

Fig. 7 Experiment 4 (Impact of parameter): impact of k in the clustering scheme; for clustering closeness, the smaller the better; for clustering correctness, the larger the better.

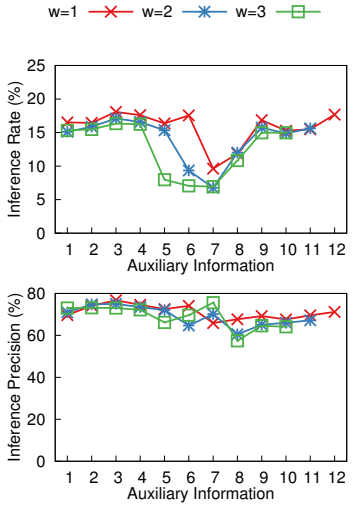


Fig. 8 Experiment 5 (Attack effectiveness): severity of clustering-based attack in VM dataset.

表 7-2 Experiment 6 (Security implications)

(a) Distribution-based attacks

Type	Extension Name	Range of File Size	Raw Inference Rate	
			Distribution-o	Distribution^S-o
Office	doc(x), ppt(x), xls(x)	10KB-1MB	4.9%	5.3%
Picture	jpg, png	10-100KB	7.7%	6.7%
Source	c, h, cpp, java, py	10-20KB	17.1%	15.0%
Database	db, po	20-700KB	2.6%	2.4%
Disk	vmdk, img	200MB-1GB	15.8%	16.7%

(b) Clustering-based attack

Users	Raw Inference Rate
1	13.2%
2	22.4%
3	25.7%
4	28.4%
5	18.5%
6	30.8%

7.3.0.0.2 Experiment 5 (Attack effectiveness): We now study the effectiveness of the clustering-based attack. Due to the boundary shift of fixed-size segment, it has low effectiveness (about 1% inference rate in our test) against the FSL dataset. Thus, we use the VM dataset to examine its severity. To configure the attack, we set k = 0.8 for clustering, and (u, r, t) = (5000, 100, 0.5) for relating ciphertext clusters to corresponding plaintext clusters.

In our micro-benchmarking, we find that the chunk-level inference in the clustering-based attack only infers thousands of chunks correctly, which contributes to a negligible inference rate (e.g., less than 0.01%). Thus, we focus on the segment-level inference, which presents the bottom line of severity in the clustering-based attack. To be consistent with the chunk-level measurements, we count inference rate and precision based on the unique chunks in each correctly inferred segment.

We use the same evaluation methodology of Experiment 3, and report the results in Figure 8. Specifically, the x-axis describes the index i (where $1 \leq i \leq 12$) of the VM backup that is used as the auxiliary information for attack, while the y-axis presents the average inference rate or precision of all VM users against the $(i + w)$th backup (where w = 1, 2, and 3). We observe that both the inference rate and the inference precision fluctuate significantly. For example, when using the 3rd backup as the auxiliary information, the attack achieves the highest inference rates at 18.1%, 17.1% and 16.3%, with the precisions of 76.8%, 75.0% and 73.0% for w = 1, 2, and 3, respectively. On the other hand, when using the 7th backup as the auxiliary information, the corresponding inference rates and precisions drop down to 9.6% and 65.9%, 6.8% and 71.2%, and 6.9% and 75.6%, respectively. The reason is that the VM users have heavy updates after the 7th week, and this reduces the correlation of adjacent backups. On average, for w = 1, 2 and 3, the clustering-based attack infers 15.8%, 14.0%, and 12.6% ciphertext-plaintext pairs, with the precisions of 71.1%, 69.1%, and 68.9%, respectively.

7.4 Results of Security Implications

We thus far have examined the severity of inference attacks by quantifying the correctly inferred ciphertext-plaintext pairs. However, it remains an open issue that what are the security implications informed by these results and how the frequency analysis attacks bring actual damage. In the following experiment, we evaluate the security implications of our attacks based on the *raw inference rate*, defined as the percentage of raw data content

affected by correctly inferred chunks.

7.4.0.0.1 Experiment 6 (Security implications): We first consider the distribution-based attack, and evaluate its raw inference rate against *different types* of files. We drive the evaluation using the FSL dataset, since only the FSL dataset includes file metadata (that includes the extension names of files) in plaintext. Specifically, we focus on five types of files that have specific extension names (see Table 7-2(a)): office files (*Office*), picture files (*Picture*), programming source files (*Source*), database files (*Database*), and disk image files (*Disk*). These files occupy more than 60% of raw content of FSL snapshots.

We apply the methodology of Experiment 2, and evaluate the raw inference rates of Distribution^{S}-o and Distribution-o. Table 7-2(a) shows the results. Both attack instances have high raw inference rates against *Disk* (e.g., 15.8% for Distribution-o and 16.7% for Distribution^{S}-o), because each disk file includes a large number of rarely updated chunks that form high locality within the same file. Interestingly, we observe that *Source*, although each file is of a small size, also incurs high raw inference rates by the distribution-based attacks (e.g., 17.1% for Distribution-o and 15.0% for Distribution^{S}-o). The reason is that programming source files are often stored together in file system (e.g., the source files that belong to the same project locate in an identical directory) and form a large stretch of correlated chunks, which present high locality across files. For small and scattered files (e.g., *Office*, *Picture*, and *Database*), the distribution-based attacks have relative low raw inference rates.

Observation (8) – *The severity of the distribution-based attack depends on the update frequencies, sizes, and spatiality of target files.*

We examine the security implication of the clustering-based attack using the VM dataset. Specifically, we use the 11th backup of each VM user to infer original content in corresponding 13th VM backup. Since the VM dataset does not contain any metadata, we count the raw inference rate based on the whole data content in each VM snapshot. Note that we filter all zero chunks in the count of raw inference rate, because they occupy a large fraction in VM disk images [6].

We use the same configuration of Experiment 5, and evaluate raw inference rate based on segment-level inference. Table 7-2(b) shows the results for different users. We observe that the clustering-based attack achieves high severity against the VM dataset. For example, it infers up to 30.8% raw content of User6's VM backup. On average, the raw inference rate of all users is as high as 23.2%.

Observation (9) – *The clustering-based attack threatens the confidentiality of VM disk images.*

第八章 Countermeasure Discussion

In this section, we discuss the advantages and disadvantages of possible countermeasures against the leakage channels exploited in this paper. Note that the countermeasures against size leakage are not enough for preventing our attacks, in which the size information is just optional.

8.0.0.0.1 Against frequency leakage: MinHash encryption [22,27] encrypts each plaintext with a key derived from the minimum chunk hash over a set of its adjacent chunks, and possibly maps identical plaintexts into different ciphertexts. The rationale for defense is that MinHash encryption changes frequencies, and disturbs frequency ranking. Note that MinHash encryption also prevents the frequency analysis attacks in this paper, because we target deterministic encryption (e.g., MLE [8]), while MinHash encryption is non-deterministic and changes the frequency distribution of chunks. The disadvantage is that MinHash encryption degrades the storage saving by deduplication, since it breaks the one-to-one mapping between plaintexts and ciphertexts. In addition, MinHash encryption is not an active countermeasure, since its randomness essentially depends on the minimum chunk hashes in the target workloads.

A recent work [46] suggests intentionally adding duplicate chunks to prevent traffic analysis against client-side deduplication. The approach [46] can be applied to address the frequency analysis attacks, since it changes the frequencies of chunks. Compared with MinHash encryption, the countermeasure [46] does not degrade storage efficiency, since only duplicate chunks are added. On the other hand, it requires the priori knowledge that if particular chunks are duplicate, and only suits the client-side deduplication, which possibly introduces additional leakage channels (see Section 第九章).

Several extended MLE instantiations build on strong cryptographic primitives to defend against the frequency leakage of encrypted deduplication, such as randomized encryption that supports equality testing [47], hybrid encryption [48], and interactions with fully homomorphic encryption [49]. They provide provable security, yet how they can be implemented and deployed in practice remains unexplored.

8.0.0.0.2 Against order leakage: A simple countermeasure is to disturb the deduplication processing sequence of chunks. For example, we can operate an additional order-

scrambling process on the stream of plaintexts before encryption, so as to hide the actual logical order of each plaintext. This prevents the distribution-based attack, because the adversary cannot identify neighboring information correctly. On the other hand, it is not always effective against the clustering-based attack. If scrambling operates on just a small basis (e.g., the orders of plaintexts in each segment are scrambled), the clustering-based attack still works. The disadvantage of the order-scrambling countermeasure is that it breaks chunk locality and leads to performance drop in large-scale deduplication [1, 2, 23].

8.0.0.0.3 Against size leakage: As suggested by prior work [40], we can pad each plaintext with additional data to obfuscate the actual size of this plaintext. However, we note that the padding scheme is not straightforward, since it requires to add identical redundancies to the same plaintexts; otherwise the storage system cannot detect duplicates for deduplication. One possible solution is to build on the paradigm of MLE [8, 50]. We compute the cryptographic hash of each plaintext as a seed, and use it to generate variable-size pseudorandom data to be padded with the plaintext. Like MLE, this solution comes at the expense of server-aided approach [50] to defend against brute-force attack (see Section 第九章).

Instead of variable-size chunking, we can use fixed-size chunking to generate equal-size chunks for deduplication. Since all chunks have the same size, the adversary cannot exploit size information to differentiate them. Although fixed-size chunking suffers from boundary shift, it achieves almost the same storage saving of deduplication as variable-size chunking in VM disk images [6]. Thus, we suggest applying fixed-size chunking in its favorable workloads to prevent size leakage.

第九章 Related Work

MLE instantiations: Recall from Section 第三章 that MLE [8] formalizes the cryptographic foundation of encrypted deduplication. The first published MLE instantiation is convergent encryption (CE) [29], which uses the cryptographic hash of a plaintext and its corresponding ciphertext as the MLE key and the tag, respectively. Other CE variants include: hash convergent encryption (HCE) [8], which derives the tag from the plaintext while still using the hash of the plaintext as the MLE key; random convergent encryption (RCE) [8], which encrypts a plaintext with a fresh random key to form a non-deterministic ciphertext, protects the random key by the MLE key derived from the hash of the plaintext, and attaches a deterministic tag derived from the plaintext for duplicate checks; and convergent dispersal (CD) [26], which extends CE to secret sharing by using the cryptographic hash of a plaintext as the random seed of a secret sharing algorithm. Since all the above instantiations derive MLE keys and/or tags from the plaintexts *only*, they are vulnerable to the offline brute-force attack [50] if the plaintext is *predictable* (i.e., the number of all possible plaintexts is limited) , as an adversary can exhaustively derive the MLE keys and tags from all possible plaintexts and check if any plaintext is encrypted to a target ciphertext (in CE, HCE, and CD) or mapped to a target tag (in RCE). The brute-force attack has been demonstrated to learn file information [**?**].

To protect against the offline brute-force attack, DupLESS [50] implements server-aided MLE by managing MLE keys in a standalone key server, which ensures that each MLE key cannot be derived from a message offline. DupLESS employs two mechanisms to achieve robust MLE key management: (i) oblivious key generation, in which a client always obtains a deterministic MLE key for a message from the key server without revealing the message content to the key server, and (ii) rate limiting, which limits the key generation requests from the client and prevents the online brute-force attack. Other studies extend server-aided MLE to address various aspects, such as reliable key management [51], transparent pricing [52], peer-to-peer key management [53], and rekeying [27]. However, server-aided MLE still builds on deterministic encryption. To our knowledge, existing MLE instantiations (based on either CE or server-aided MLE) are all vulnerable to the inference attacks studied in this paper.

9.0.0.0.1 Attacks against (encrypted) deduplication: In addition to the offline brute-force attack, previous studies consider various attacks against deduplication storage, and such attacks generally apply to encrypted deduplication as well. For example, the side-channel attack [7, 34] enables adversaries to exploit the deduplication pattern to infer the content of uploaded files from target users or gain unauthorized access in client-side deduplication; it is shown that the side-channel attack (and other related attacks) was successfully launched against Dropbox in 2010 [35]. The duplicate-faking attack [8] compromises message integrity via inconsistent tags. Ritzdorf *et al.* [40] exploit the leakage of the chunk size to infer the existence of files. The locality-based attack [22] exploits frequency analysis to infer ciphertext-plaintext pairs. Our work follows the line of work on inference attacks [22, 40], yet provides a more in-depth study of inference attacks against encrypted deduplication via various types of leakage.

9.0.0.0.2 Defense mechanisms: Section 第八章 discusses the countermeasures agasint the frequency, order and size leakage of encrypted deduplication. Additional defense mechanisms are designed to protect against other types of attacks. As mentioned above, server-aided MLE [50] can defend against the offline brute-force attack. Server-side deduplication [7,26,54] and proof-of-ownership [34,36,37] can defend against the side-channel attack. Server-side tag generation [29, 50] and guarded decryption [8] can defend against the duplicate-checking attack.

9.0.0.0.3 Inference attacks: Several inference attacks have been proposed against encrypted databases [**?**, 12–16] and keyword search [17–21]. They all exploit the deterministic encryption nature to identify different types of leakage. Our work differs from them by specifically focusing on frequency analysis against encrypted deduplication.

第十章 Conclusion

Encrypted deduplication applies deterministic encryption, and leaks the frequencies of plaintexts. This paper revisits the security vulnerability due to frequency analysis, and demonstrates that encrypted deduplication is even more vulnerable towards inference attacks. We propose two new frequency analysis attacks, both of which achieve high inference rate and high inference precision, while under different assumptions of adversarial knowledge. We empirically evaluate our attacks with three real-world datasets, present a variety of new observations about their natures, and further analyze how they bring actual damages. We also discuss the advantages and disadvantages of possible countermeasures to advise practitioners for securely implementing and deploying encrypted deduplication storage systems.

We pose three directions for future work. First, we consider a complete old backup as the auxiliary information, and do not study how to launch attack if the adversary only has partial knowledge about the old backup. Possibly, we can still apply the frequency analysis attacks to extract characteristics from the available partial backup, and infer ciphertext-plaintext pairs by comparing the characteristics with those in the target backup. One direction is can we design advanced inference attacks, which perform better than directly applying our attacks in this partial knowledge case?

Second, we adjust the parameters of the frequency analysis attacks based on their effectiveness, which can only be learnt after the attacks have happened. We do not study how to derive the optimal parameters from auxiliary information beforehand. Future work may do better.

Third, we do not implement attack prototypes against real-world encrypted deduplication storage systems. Another direction is to deploy our attack design and report the vulnerability of encrypted deduplication in practice.

致 谢

参考文献

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

[2] M. Lillibridge, K. Eshghi, D. Bhagwat, et al. Sparse indexing: Large scale, inline deduplication using sampling and locality[C]. Proc. of USENIX FAST, 2009,

[3] G. Wallace, F. Douglis, H. Qian, et al. Characteristics of backup workloads in production systems[C]. Proc. of USENIX FAST, 2012,

[4] F. Douglis, A. Duggal, P. Shilane, et al. The logic of physical garbage collection in deduplicating storage[C]. Proc. of USENIX FAST, 2017,

[5] D. T. Meyer, W. J. Bolosky. A study of practical deduplication[C]. Proc. of USENIX FAST, 2011,

[6] K. Jin, E. L. Miller. The effectiveness of deduplication on virtual machine disk images[C]. Proc. of ACM SYSTOR, 2009,

[7] D. Harnik, B. Pinkas, A. Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage[J]. IEEE Security & Privacy, 2010, 8(6): 40-47

[8] M. Bellare, S. Keelveedhi, T. Ristenpart. Message-locked encryption and secure deduplication[C]. Proc. of EUROCRYPT, 2013,

[9] D. X. Song, D. Wagner, A. Perrig. Practical techniques for searches on encrypted data[C]. Proc. of IEEE S&P, 2000,

[10] A. Boldyreva, N. Chenette, Y. Lee, et al. Order-preserving symmetric encryption[C]. Proc. of EUROCRYPT, 2009,

[11] I. A. Al-Kadit. Origins of Cryptology: The Arab Contributions[J]. Cryptologia, 1992, 16(2): 97-126

[12] M.-S. Lacharite, B. Minaud, K. G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage[C]. Proc. of IEEE S&P, 2018,

[13] P. Grubbs, K. Sekniqi, V. Bindschaedler, et al. Leakage-abuse attacks against order-revealing encryption[C]. Proc. of IEEE S&P, 2017,

[14] G. Kellaris, G. Kollios, K. Nissim, et al. Generic attacks on secure outsourced databases[C]. Proc. of ACM CCS, 2016,

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

[16] F. B. Durak, T. M. DuBuisson, D. Cash. What else is revealed by order-revealing encryption[C]. Proc. of ACM CCS, 2016,

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

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

[19] D. Cash, P. Grubbs, J. Perry, et al. Leakage-abuse attacks against searchable encryption[C]. Proc. of ACM CCS, 2015,

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

[21] P. Grubbs, R. McPherson, M. Naveed, et al. Breaking web applications built on top of encrypted data[C]. Proc. of ACM CCS, 2016,

[22] J. Li, C. Qin, P. P. C. Lee, et al. Information leakage in encrypted deduplication via frequency analysis[C]. Proc. of IEEE/IFIP DSN, 2017,

[23] W. Xia, H. Jiang, D. Feng, et al. Silo: A similarity locality based near exact deduplication scheme with low ram overhead and high throughput[C]. Proc. of USENIX ATC, 2011,

[24] D. Bhagwat, K. Eshghi, D. D. E. Long, et al. Extreme binning: Scalable, parallel deduplication for chunk-based file backup[C]. Proc. of IEEE MASCOTS, 2009,

[25] Z. Sun, G. Kuenning, S. Mandal, et al. A long-term user-centric analysis of deduplication patterns[C]. Proc. of IEEE MSST, 2016,

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

[27] C. Qin, J. Li, P. P. C. Lee. The design and implementation of a rekeying-aware encrypted deduplication storage system[J]. ACM Transactions on Storage, 2017, 13(1): 9:1-9:30

[28] J. Black. Compare-by-hash: A reasoned analysis[C]. Proc. of USENIX ATC, 2006,

[29] J. R. Douceur, A. Adya, W. J. Bolosky, et al. Reclaiming space from duplicate files in a serverless distributed file system[C]. Proc. of IEEE ICDCS, 2002,

[30] A. Adya, W. J. Bolosky, M. Castro, et al. Farsite: Federated, available, and reliable storage for an incompletely trusted environment[C]. Proc. of USENIX OSDI, 2002,

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

[32] M. W. Storer, K. Greenan, D. D. E. Long, et al. Secure data deduplication[C]. ACM StorageSS, 2008,

[33] P. Anderson, L. Zhang. Fast and secure laptop backups with encrypted de-duplication[C]. Proc. of USENIX LISA, 2010,

[34] S. Halevi, D. Harnik, B. Pinkas, et al. Proofs of ownership in remote storage systems[C]. Proc. of ACM CCS, 2011,

[35] M. Mulazzani, S. Schrittwieser, M. Leithner, et al. Dark clouds on the horizon: Using cloud storage as attack vector and online slack space[C]. Proc. of USENIX Security, 2011,

[36] J. Xu, E.-C. Chang, J. Zhou. Weak leakage-resilient client-side deduplication of encrypted data in cloud storage[C]. Proc. of ACM ASIACCS, 2013,

[37] R. D. Pietro, A. Sorniotti. Boosting efficiency and security in proof of ownership for deduplication[C]. Proc. of ACM ASIACCS, 2012,

[38] A. Juels, B. S. Kaliski, Jr.. Pors: Proofs of retrievability for large files[C]. Proc. of ACM CCS, 2007,

[39] G. Ateniese, R. Burns, R. Curtmola, et al. Provable data possession at untrusted stores[C]. Proc. of ACM CCS, 2007,

[40] H. Ritzdorf, G. O. Karame, C. Soriente, et al. On information leakage in deduplicated storage systems[C]. Proc. of ACM CCSW, 2016,

[41] R. Kumar, J. Novak, B. Pang, et al. On anonymizing query logs via token-based hashing[C]. Proc. of ACM WWW, 2007,

[42] Q. A. Wang. Probability distribution and entropy as a measure of uncertainty[J]. Journal of Physics A: Mathematical and Theoretical, 2008, 41(6):

[43] A. Broder. On the resemblance and containment of documents[C]. Proc. of ACM SEQUENCES, 1997,

[44] S. C. Johnson. Hierarchical clustering schemes[J]. Psychometrika, 1967, 32(3): 241-254

[45] Z. Sun, G. Kuenning, S. Mandal, et al. Cluster and single-node analysis of long-term deduplication patterns[J]. ACM Transactions on Storage, 2018, 14(2): 13:1-13:25

[46] P. Zuo, Y. Hua, C. Wang, et al. Mitigating traffic-based side channel attacks in bandwidth-efficient cloud storage[C]. Proc. of IEEE IPDPS, 2018,

[47] M. Abadi, D. Boneh, I. Mironov, et al. Message-locked encryption for lock-dependent messages[C]. Proc. of CRYPTO, 2013,

[48] J. Stanek, A. Sorniotti, E. Androulaki, et al. A secure data deduplication scheme for cloud storage[C]. Proc. of FC, 2014,

[49] M. Bellare, S. Keelveedhi. Interactive message-locked encryption and secure deduplication[C]. Proc. of PKC, 2015,

[50] M. Bellare, S. Keelveedhi, T. Ristenpart. DupLESS: Server-aided encryption for deduplicated storage[C]. Proc. of USENIX Security, 2013,

[51] Y. Duan. Distributed key generation for encrypted deduplication: Achieving the strongest privacy[C]. Proc. of ACM CCSW, 2014,

[52] F. Armknecht, J.-M. Bohli, G. O. Karame, et al. Transparent data deduplication in the cloud[C]. Proc. of ACM CCS, 2015,

[53] J. Liu, N. Asokan, B. Pinkas. Secure deduplication of encrypted data without additional independent servers[C]. Proc. of ACM CCS, 2015,

[54] F. Armknecht, C. Boyd, G. T. Davies, et al. Side channels in deduplication: Trade-offs between leakage and efficiency[C]. Proc. of ACM ASIACCS, 2017,

[55] M. Bellare, A. Boldyreva, A. O'Neill. Deterministic and efficiently searchable encryption[C]. Proc. of CRYPTO, 2007,

[56] M. Bellare, T. Ristenpart, P. Rogaway, et al. Format-preserving encryption[C]. Proc. of Selected Areas in Cryptography, 2009,

[57] A. Boldyreva, N. Chenette, A. O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions[C]. Proc. of CRYPTO, 2011,

[58] E. Bosman, K. Razavi, H. Bos, et al. Dedup est machina: Memory deduplication as an advanced exploitation vector[C]. Proc. of IEEE S&P, 2016,

[59] R. Curtmola, J. Garay, S. Kamara, et al. Searchable symmetric encryption: Improved definitions and efficient constructions[C]. Journal of Computer Security, 2006,

[60] D. Gruss, D. Bidner, S. Mangard. Practical memory deduplication attacks in sandboxed javascript[C]. Proc. of ESORICS, 2015,

[61] M.-S. Lacharité, K. G. Paterson. Frequency-smoothing encryption: Preventing snapshot attacks on deterministically encrypted data[J]. IACR Transactions on Symmetric Cryptology, 2018, 2018(1): 277-313

[62] A. Muthitacharoen, B. Chen, D. Mazieres. A low-bandwidth network file system[C]. Proc. of ACM SOSP, 2001,

[63] R. A. Popa, C. M. S. Redfield, N. Zeldovich, et al. Cryptdb: Protecting confidentiality with encrypted query processing[C]. Proc. of ACM SOSP, 2011,

[64] R. A. Popa, E. Stark, J. Helfer, et al. Building web applications on top of encrypted data using mylar[C]. Proc. of USENIX NSDI, 2014,

[65] R. Popa, C. Redfield, N. Zeldovich, et al. Cryptdb: Protecting confidentiality with encrypted query processing[C]. Proc. of USENIX SOSP, 2011,

[66] P. Shah, W. So. Lamassu: Storage-efficient host-side encryption[C]. Proc. of USENIX ATC, 2015,

[67] Y. Shin, D. Koo, J. Hur. A survey of secure data deduplication schemes for cloud storage systems[J]. ACM Computing Surveys, 2017, 49(4): 1-38

[68] Z. Wilcox-O'Hearn, B. Warner. Tahoe: the least-authority filesystem[C]. Proc. of ACM StorageSS, 2008,

[69] J. Xiao, Z. Xu, H. Huang, et al. Security implications of memory deduplication in a virtualized environment[C]. Proc. of IEEE/IFIP DSN, 2013,

[70] L. L. You, K. T. Pollack, D. D. E. Long. Deep store: An archival storage system architecture[C]. Proc. of IEEE ICDE, 2005,

附　录