# Data Leak Detection As a Service

Xiaokui Shu and Danfeng (Daphne) Yao

Department of Computer Science
Virginia Tech
Blacksburg VA, USA
{subx,danfeng}@cs.vt.edu

**Abstract.** We describe a network-based data-leak detection (DLD) technique, the main feature of which is that the detection does not reveal the content of the sensitive data. Instead, only a small amount of specialized digests are needed. Our technique – referred to as the *fuzzy fingerprint* detection – can be used to detect accidental data leaks due to human errors or application flaws. The privacy-preserving feature of our algorithms minimizes the exposure of sensitive data and enables the data owner to safely delegate the detection to others (e.g., network or cloud providers). We describe how cloud providers can offer their customers data-leak detection as an add-on service with strong privacy guarantees. We perform extensive experimental evaluation on our techniques with large datasets. Our evaluation results under various data-leak scenarios and setups show that our method can support accurate detection with very small number of false alarms, even when the presentation of the data has been transformed.

**Key words:** privacy, data leak, network security, protocol

## 1 Introduction

Typical approaches to preventing data leak are under two categories – host-based solutions and network-based solutions. Host-based approaches may include *i)* encrypting data when not used [4], *ii)* detecting stealthy malware with anti-virus scanning or monitoring the host [29, 31, 18], and *iii)* enforcing policies to restrict the transfer of sensitive data. These approaches are complementary and can be deployed simultaneously.

We present a *network-based* data-leak detection (DLD) solution that complements host-based methods. Network-based data-leak detection focuses on analyzing unencrypted outbound network traffic through *i)* deep packet inspection or *ii)* information theoretic analysis (e.g., through entropy analysis [13]). For the deep packet inspection approach, a straightforward solution requires inspecting every packet for the occurrence of any of the sensitive data defined in the sensitive database. Such solutions generate alerts if the sensitive data is found in the outgoing traffic. However, this simple solution requires storing sensitive data in *plaintext* in the detection system.

The reason that this plaintext-based comparison mechanism is undesirable is two-fold: *i)* the machine performing the comparison may be compromised, which

reveals sensitive data [1], and *ii)* it does not support the outsource of data-leak detection operations, as the provider performing the DLD service may learn or accidentally expose the sensitive data. In addition to provide the regular networking, computing, or storage services, network or cloud providers may introduce additional security protection for their customers. For their customers, these add-on security services – such as data-leak detection – are attractive, as they may have a lower cost compared to building in-house security management of their own. Thus, one may outsource the data-leak detection to a DLD provider. However, the data owner may not allow the DLD provider to access the sensitive data. *The technical challenge is that the detection algorithm needs to provide guarantees on the secrecy of customers' sensitive data while still enabling the provider to identify signs of data leak in the traffic.*

This problem of the lack of support for privacy-enhancing data-leak detection has not been systematically addressed in the security literature. In this paper we design, implement, and experimentally evaluate an efficient technique that enhances the data privacy during the data-leak detection operations. Our method is based on a fast and practical one-way computation and does not require any expensive cryptographic operations. We provide extensive experimental evidences and theoretical analysis to demonstrate the feasibility and effectiveness of our approach.

We model the DLD provider as an honest-but-curious (aka semi-honest) adversary. The DLD provider is trusted to perform inspection on network traffic, but may attempt to learn the information about the sensitive database provided by the data owner, or to discover the leaked data easily from the network traffic. Existing work on cryptography-based multi-party computation is not efficient enough for practical data leak inspection in this setting. We design, implement, and evaluate a new privacy-enhancing data-leak detection system that enables the data owner to securely delegate the traffic-inspection task to DLD providers without exposing the sensitive data. It is hard for a DLD provider to learn the *exact* value of sensitive data during the detection process.

In our model, the data owner computes a special set of digests or fingerprints from the sensitive data, and then discloses only a small amount of digest information to the DLD provider. These fingerprints have important properties, which prevent the provider from gaining knowledge of the sensitive data, while they enable accurate comparison and detection. The DLD provider performs deep packet inspection to identify whether these fingerprint patterns exist in the outbound traffic of data owner's organization or not. We perform extensive experiments with real-world datasets in various data-leak scenarios to confirm the accuracy and efficiency of our proposed solutions. Our contributions are summarized as follows.

1. We describe a privacy-preserving data-leak detection (DLD) model for preventing inadvertent data leak in network traffic. Such a model yields a powerful and delegatable data-leak detection framework. For example, in the

[1] Sensitive data may be in encrypted storage, but is plaintext when in memory for comparison.

cloud computing environment the cloud provider can perform data-leak detection as an add-on service to its clients. We describe a quantitative privacy model needed for data-leak detection as a service.
We design, implement, and evaluate a new and efficient technique, *fuzzy fingerprint*, for realizing privacy-preserving data-leak detection. Fuzzy fingerprints are special digests of the sensitive data that the data owner releases to the DLD provider. We describe the operations in our protocol that is run between the data owner and the DLD provider.

2. We implement our detection system and perform extensive experimental evaluation on 2.6 GB Enron dataset, Internet surfing traffic of 20 users, and also 5 simulated real-world data-leak scenarios to measure the privacy guarantee, detection rate and efficiency of our technique. Our results indicate high accuracy performed by our underlying scheme with very low false positive rate. It also shows that the detection accuracy does not degrade when only partial (sampled) sensitive-data digests are used. In addition, these partial fingerprints represent the full set of data without any bias.

The rest of the paper is organized as follows. Our models and design requirements for a privacy-preserving data-leak detection system are presented next. Details of our system including digest computation, data-inspection strategies are described in Section 3. We analyze the privacy in Section 4, and also point out the limitations of our method. Our implementation and evaluation are described in Section 5. Related work is given in Section 6. Conclusions and future work are given in Section 7.

# 2 Model and Overview

There is a privacy goal and threat model beside the normal security goal and threat model for any solution to outsource data-leak detection. The former is for preventing the service provider from gaining knowledge about the sensitive data during the detection, whereas the latter relates to preventing unauthorized transmission of sensitive data. There are two types of players in our model: the organization (i.e., data owner) and the data-leak detection (DLD) provider.

- *Organization* owns the sensitive data and authorizes the DLD provider to inspect the network traffic from the organizational networks for anomalies, namely inadvertent data leak. However, the organization does not want to directly reveal the sensitive data to the provider.
- *DLD provider* inspects the network traffic for potential data leaks. The inspection can be performed offline without causing any real-time delay in routing the packets. However, the DLD provider may attempt to gain knowledge about the sensitive data.

We describe the security and privacy goals in Section 2.1 and Section 2.2.

## 2.1 Security Goal and Threat Model

We categorize three causes for sensitive data to appear on the outbound traffic of an organization, including the legitimate data use by the employees.

- Case I *Inadvertent data leak*: The sensitive data is accidentally leaked in the outbound traffic by a legitimate user. This paper focuses on detecting this type of accidental data leaks over the network. Inadvertent data leak may be due to human errors such as forgetting to use encryption, carelessly forwarding an internal email and attachments to outsiders without encryption, or due to application flaws (such as described in [19]).
- Case II *Malicious data leak*: A rogue insider or malicious and stealthy software may steal sensitive personal or organizational data from a host. Because the malicious adversary can use strong encryption or steganography to disable content-based traffic inspection, thus this type of leaks (including covert channels) are out of the scope of our network-based solution. Host-based defenses (such as detecting the infection onset [33]) need to be deployed instead.
- Case III *Legitimate and intended data transfer*: The sensitive data is sent by a legitimate user intended for legitimate purposes. In this paper, we assume that legitimate data transfers use data encryption such as SSL, which allows one to distinguish it from the inadvertent data leak. Therefore, in what follows we assume that plaintext sensitive data appearing in network traffic is only due to inadvertent data leaks.

The security goal in this paper is to detect the inadvertent data leak in Case I. In this scenario, the traffic is usually not encrypted and thus deep packet inspection is feasible. Network-based security approaches are not effective against data leak caused by malware or rogue insiders as in Case II, because the intruder may use strong encryption when transmitting the data.

## 2.2 Privacy Goal and Threat Model

To prevent the DLD provider from gaining knowledge of the sensitive data during the detection process, we need to set up a privacy goal that is complementary to the security goal above. We model the DLD provider as a semi-honest adversary, who follows our protocol to carry out the operations, but may attempt to gain knowledge about the sensitive data of the data owner. Our privacy goal is defined as follows. The DLD provider is given digests of sensitive data from the data owner and the content of network traffic to be examined. The DLD provider should not find out the exact value of a piece of sensitive data with more than $\frac{1}{K}$ probability, where $K$ is an integer representing the number of all possible sensitive-data candidates that can be inferred by the DLD provider.

We present a novel privacy-preserving DLD model with a new fuzzy fingerprint mechanism to improve the data protection against semi-honest DLD provider. We generate digests of sensitive data through a one-way function, and then hide the sensitive values among other non-sensitive values via fuzzification, The privacy guarantee is much higher than $\frac{1}{K}$ when there is no leak in traffic, because the adversary's inference can only be done through brute-force guesses.

The traffic content is accessible by the DLD provider in plaintext. Therefore, in the event of true data leak, the DLD provider may learn about the leaked information, which is inevitable for all deep-packet inspection approaches. Our

unique solution confines the amount of maximally information learned during the detection and provides quantitative guarantee for the data privacy.

### 2.3 Overview of Privacy-Enhancing DLD

Our privacy-preserving data-leak detection method supports practical data-leak detection as a service and minimizes the knowledge that a DLD provider may gain during the process. Figure 1 illustrates the six operations between the data owner and the DLD provider in our protocol, which include Preprocess run by the data owner to prepare the digests of sensitive data, Release for the data owner to send the digests to the DLD provider, Monitor and Detect for the DLD provider to collect outgoing traffic of the organization, compute digests of traffic content, and identify potential leaks, Report for the DLD provider to return data leak alerts to the data owner where there may be false positives (i.e., false alarms), and Postprocess for the data owner to pinpoint true data leak instances. We explain the operations in details in the next section.

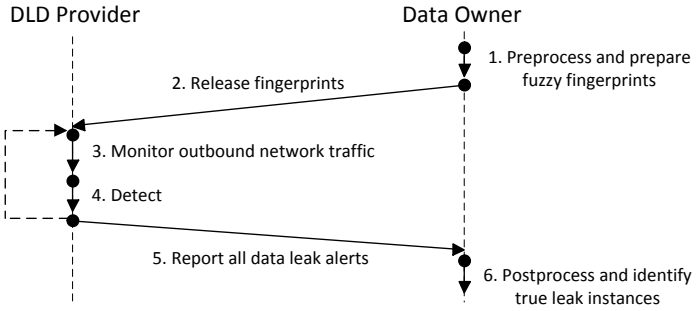


**Fig. 1.** Privacy-preserving DLD Model

The protocol is based on strategically computing data similarity, specifically the quantitative similarity between the sensitive information and the observed network traffic. High similarity indicates potential data leak. For data-leak detection, the ability to tolerate certain degree of data transformation in traffic is important. We refer to this property as *noise tolerance*. Our key idea for fast and noise-tolerant comparison is the design and use of a set of *local features* that are representative of local data patterns. Local features preserve data patterns even when modifications (insertion, deletion, and substitution) are made to parts of the data. To achieve the privacy requirement, the data owner generates a special type of digests, which we call fuzzy fingerprints. Intuitively, the purpose of fuzzy fingerprints is to hide the true sensitive data in the crowd so that the DLD provider is unable to learn its exact value. We describe the technical details next.

## 3 Fuzzy Fingerprint Method and Protocol

We describe technical details of our fuzzy fingerprint mechanism in this section.

### 3.1 Fingerprints

The DLD provider obtains digests of sensitive data from the data owner. The data owner uses Rabin fingerprint algorithm [24] and a sliding window to generate short and hard-to-reverse (i.e., oneway) digests through the fast polynomial modulus operation. Rabin fingerprints are computed as polynomial modulus operations, and can be implemented with fast XOR, shift, and table look-up operations. It has a unique min-wise independence property [7], which allows randomly sampling of the digests without creating any bias.

The shingle-and-fingerprint process is defined as follows. For a binary string, we first generate $q$-grams (shingles) using a sliding window, and then compute Rabin fingerprint of each $k$-bit shingle using irreducible polynomial $p(x)$:

$$f_1 = c_1 x^{k-1} + c_2 x^{k-2} + \ldots + c_{k-1} x + c_k \bmod p(x)$$

From the detection respective, a straightforward method is for the DLD provider to raise an alert if any sensitive fingerprint matches the fingerprints generated from the traffic. However, this approach has a privacy issue. In case of a data leak detected, there is a match between two fingerprints from sensitive data and network traffic. Then, the DLD provider learns the corresponding shingle, as it knows the content of the packet. Therefore, the central challenge is *to prevent the DLD provider from learning the sensitive values even in data-leak scenarios*, while allowing the provider to carry out the traffic inspection.

We propose a novel and efficient technique to address this problem. The main idea is to relax the comparison criteria by strategically introducing matching instances on the DLD provider's side *without increasing false alarms for the data owner*. Specifically, *i)* the data owner perturbs the sensitive-data fingerprints before disclosing them to the DLD provider, and *ii)* the DLD provider detects leaking by a range-based comparison instead of the exact match. The range used in the comparison is pre-defined by the data owner and correlates to the perturbation procedure. We first define the *fuzzy length* and *fuzzy set* next and then describe how they are used in our detailed protocol in Section 3.2.

**Definition 1.** *Given a fingerprint $f$, fuzzy length $p_d$ ($p_d < p_f$) is the number of the least significant bits in $f$ that may be perturbed by the data owner.*

**Definition 2.** *Given a fuzzy length $p_d$, and a collection of fingerprints, the fuzzy set $S_{f,p_d}$ of a fingerprint $f$ is the set of fingerprints in the collection whose values differ from $f$ by at most $2^{p_d} - 1$.*

In Definition 1 for fuzzy length, $p_f$ denotes the total length of a fingerprint. In Definition 2, the size of the fuzzy set $|S_{f,p_d}|$ is upper bounded by $2^{p_d}$, but the actual size may be smaller due to the sparsity of the fingerprint space.

### 3.2 Operations in Our Protocol

1. Preprocess:
   This operation is run by the data owner on some sensitive dataset. The data owner chooses the public parameters $(k, p(x), p_d)$, where $k$ is the length of

shingles, $p(x)$ is an irreducible polynomial for computing Rabin fingerprint, and $p_d$ is the fuzzy length. The length of a fingerprint is denoted by $p_f$ [2]. The data owner first computes the set $\mathbb{S}$ of Rabin fingerprints of the sensitive data. Then the data owner transforms each fingerprint $f \in \mathbb{S}$ into a fuzzy fingerprint $f^*$ as follows. Given the fingerprint $f$ of some shingle $v$ and a fuzzy length $p_d$, the data owner flips an unbiased coin $p_d$ times to generate the new least significant $p_d$ bits in $f$. The rest of the bits in $f$ are unchanged. The transformation generates a fuzzy fingerprint $f^*$ of $f$. We denote the resulting set of fuzzy fingerprints by $\mathbb{S}^*$, which is the output of this operation.

2. Release:
   This operation is run by the data owner. The fuzzy fingerprint set $\mathbb{S}^*$ obtained from the Preprocess operation above is released to the DLD provider for use in the detection, along with the public parameters $(k, p(x), p_d)$. The real fingerprint $f$ and the corresponding sensitive shingle $v$ are kept at the data owner and not released to the DLD provider.
3. Monitor: This operation is run by the DLD provider. The DLD provider monitors the network traffic $\mathbb{T}$ from the data owner's organization. The header of the packet in $\mathbb{T}$ is removed and the payload is collected. The processed traffic $\tilde{\mathbb{T}}$ is the output.
4. Detect:
   This operation is run by the DLD provider on $\tilde{\mathbb{T}}$ as follows.
   a) The DLD provider first computes the Rabin fingerprints of traffic content $\tilde{\mathbb{T}}$ based on the public parameters.
   b) For each fuzzy fingerprint $f^* \in \mathbb{S}^*$ of some sensitive data, and each fingerprint $f' \in \tilde{\mathbb{T}}$ from the traffic, and the public parameters, the DLD provider outputs 1 (indicating possible data leak) if values of $f^*$ and $f'$ differ by at most $2^{p_d} - 1$, and 0 otherwise.
   c) For all the data-leak matching instances detected during this range-based detection, the DLD provider records the set of $\{(x_1, f_1), \ldots, (x_i, f_i), \ldots)\}$ pairs, where $x_i$ is the shingle appearing in the traffic, and $f_i$ is its Rabin fingerprint. The DLD provider and the data owner may agree upon certain aggregation methods and a threshold for logging alerts, which we discuss more in the evaluation section 5.

   Because the fuzzy set of $f^*$ includes the original fingerprint $f$, thus the true data leak can be detected (i.e., true positive). Yet, due to the increased detection range, multiple values in the fuzzy set may trigger alerts. Because the fuzzy set is large for the given network flow, the DLD provider has a low probability of pinpointing the sensitive data, which can be bounded as shown in Section 4.
5. Report:
   The DLD provider reports the set of detected candidate leak instances $\{(x_1, f_1), \ldots, (x_i, f_i), \ldots)\}$ tuples to the data owner.
6. Postprocess:

[2] The degree of polynomial $p(x)$ is $p_f + 1$.

This operation is run by the data owner. Given the data-leak instance candidates in the reported set of tuples $\{(x_1, f_1), (x_2, f_2), \ldots\}$, the data owner searches to see if any sensitive fingerprint $f \in \mathbb{S}$ exists in the report. If there exist $f_i = f$ and $x_i = v$, i.e., the shingle $x_i$ and fingerprint $f$ in the traffic match those $(v, f)$ of the sensitive data, then there is a true data leak, otherwise the submitted candidates can be safely ignored by the data owner.

The DETECT operation can be performed between $\tilde{\mathbb{T}}$ and $\mathbb{S}^*$ via set intersection test (e.g. Formula 2 in Section 5 as one realization). The advantage of our method is that the additional matching instances introduced by fuzzy fingerprints protect the sensitive data from the DLD provider; yet they do not cause additional false alarms for the data owner, as the data owner can quickly distinguish true and false leak instances. Given the digest $f$ of a piece of sensitive data, a large collection $T$ of traffic fingerprints, and a positive integer $K \ll |T|$, the data owner can choose a fuzzy length $p_d$ such that there are at least $K-1$ other distinct digests in the fuzzy set of $f$, assuming that the shingles corresponding to these $K$ digests are equally likely to be candidates for sensitive data and to appear in network traffic. A tight fuzzy length (i.e., the smallest $p_d$ value satisfying the privacy requirement) is important for efficient POSTPROCESS operation. Due to the dynamic nature of network traffic, $p_d$ needs to be estimated accordingly. We provide quantitative analysis on fuzzy fingerprint including empirical results on different sizes of fuzzy set.

### 3.3 Extensions

*Fingerprint Filter* We develop this extension to use Bloom filter in the DETECT operation for efficient set intersection test. Bloom filter is a well-known space-saving data structure for performing set-membership test, and the range-based comparison in the DETECT operation can be generalized to the membership test with it. Bloom filter in combination with Rabin fingerprint is referred to by us as the *fingerprint filter*. We have implemented, evaluated, and compared this technique in our experiments in Section 5.

*Bit Mask* We can generalize the PREPROCESS operation with a bit mask, which specifies any arbitrarily chosen bits or any mapped bit pattern for comparison. Details of how bit mask works are discussed in our technical report [28].

*Sampling* Using the min-wise independent property of Rabin fingerprint, the data owner may *sample* the fingerprints and only reveals a subset of sensitive-data's fingerprints to the DLD provider. That is, the data owner may release a subset of $\mathbb{S}^*$ to the DLD provider in RELEASE operation. The purpose of sampling is two-fold: to increase the scalability of the comparison in the DETECT operation, and to reduce the exposure of data to the DLD provider for privacy. The subset is selected by choosing the subset of smallest fingerprints when Rabin fingerprint is equipped. More description can be found in our technical report [28].

## 4 Analysis and Discussion

We analyze the security and privacy guarantees provided by our data-leak detection system, as well as discuss the sources of possible false negatives – data

leak cases being overlooked and false positives – legitimate traffic misclassified as data leak in the detection.

*Privacy Analysis* Our privacy goal is to prevent the DLD provider from inferring the exact knowledge of all sensitive data, both the outsourced sensitive data and the matched digests in network traffic. We quantify the probability for the DLD provider to infer the sensitive shingles. Suppose there are matches between sensitive fingerprints and traffic fingerprints. Given a fuzzy length, there are multiple (e.g., $K$) fingerprints (including the sensitive data's fingerprint) that may trigger alerts at the DLD provider; thus, the DLD provider is unable to pinpoint which alerts are true data leaks. Therefore, even if sensitive data appeared on the traffic due to inadvertent data leak, the DLD provider has no more than $\frac{1}{K}$ probability of inferring the sensitive data, assuming that the shingles associated with the fuzzy set are equally likely to be sensitive data and appear in the network traffic. The size of fuzzy set $K$ is upper bounded by $2^{p_d}$. For a large shingle set of size $2^{p_f-p_d} \leq n \leq 2^{p_f}$, the expected value of $K = \frac{n}{2^{p_f}} \times 2^{p_d}$, assuming that the fingerprints of shingles are uniformly distributed. It is a reasonable assumption, especially when binary sensitive data is included, which expands the small distinguishable text space to the vast more well-distributed whole binary space. This privacy guarantee protects the sensitive data in the *worst-case* scenario.

If there is no match between sensitive and traffic fingerprints, then the adversarial DLD provider needs to brute force to reverse the Rabin fingerprinting computation to obtain the input shingle. The time needed depends on the size of shingle space. This brute-force attack is difficult for a polynomial-time adversary and thus the success probability is not included in Theorem 1. We summarize the above privacy analysis in the following theorem.

**Theorem 1.** *A polynomial-time adversary has no greater than* $\frac{2^{p_f-p_d}}{n}$ *probability of correctly inferring a sensitive shingle, where* $p_f$ *is the length of a fingerprint in bits,* $p_d$ *is the fuzzy length, and* $n \in [2^{p_f-p_d}, 2^{p_f}]$ *is the size of the set of traffic fingerprints, assuming that the fingerprints of shingles are uniformly distributed and are equally likely to be sensitive and appear in the traffic.*

*Alert Rate* We qualify the rate of alerts expected in the traffic for a sensitive data entry (the fuzzified fingerprints set of a piece of sensitive data) given the following values: the total number of fuzzified sensitive fingerprints $M$, the expected traffic fingerprints set size $n$, fingerprint length $p_f$, fuzzy length $p_d$, sampling rate $p_s \in (0, 1]$, and the expected rate $\alpha$ of the leak in terms of the percentage of fingerprints in the sensitive data entry that appear in the network traffic. Based on Theorem 1, the expected alert rate $R$ can be expressed in Equation 1. It is used to derive threshold in the detection; the detection threshold should be lower than the expected rate of alerts.

$$R = \frac{\alpha p_s K M}{n} = \frac{\alpha p_s M}{2^{p_f-p_d}} \tag{1}$$

*Collisions* Collisions may be due to where the legitimate traffic happens to contain the partial sensitive-data fingerprints by coincidence. The collision may

increase with shorter shingles, or smaller numbers of partial fingerprints, and may decrease if additional features such as the order of fingerprints are used for detection. A previous large-scale information-retrieval study empirically demonstrated the low rate of this type of collisions in Rabin fingerprint [6], which is a desirable property suggesting low unwanted false alarms in our DLD setting. Collisions due to two distinct shingles generating the same fingerprint are proved to be low [5] and are negligible.

*Dynamic data* For protecting dynamically changing data such as source code or documents under constant development or keystroke data, the digests need to be continuously updated for detection, which may not be efficient or practical. We raise the issue of how to efficiently detect dynamic data with a network-based approach as an open problem to investigate by the community.

*Space of sensitive data* The space of all text-based sensitive data may be smaller than the space of all possible shingles. Yet, when including non-ASCII sensitive data (text in UTF-8 or binaries), the space of sensitive data can be significantly expanded. Thus, the assumption in Theorem 1 is practical.

*Data modification* False negatives (i.e., failure to detect data leak) may also occur due to the data being modified by the leaking application (such as insertion, deletion, and substitution). The new shingles/fingerprints may not resemble the original ones, and cannot be detected. As a result, a packet may evade the detection. In our experiments, we evaluate the impact of several types of data transformation in real world scenarios.

# 5 Experimental Evaluation

We implement our fuzzy fingerprint framework in Python (version 2.7), including packet collection, shingling, Rabin fingerprinting and fingerprint filter. Our implementation of Rabin fingerprint is based on cyclic redundancy code (CRC). We use the padding scheme mentioned in [23] to handle small inputs, and map our shingle into a sparse fingerprint space. In all experiments, the shingles are in 8-byte, and the fingerprints are in 32-bit (33-bit irreducible polynomials in Rabin fingerprint). We set up a virtual network environment in Oracle VirtualBox, simulating a scenario where the sensitive data is leaked from a local network to the Internet. Valid users' hosts (Windows 7) are put into the local network, which connects to the Internet via a gateway (Linux). The gateway dumps the network traffic and sends it to a DLD server/provider (Linux). Using the sensitive-data fingerprints defined by the users in the local network, the DLD server performs off-line data leak detection. We also set up some servers (FTP, HTTP, etc.) and a hacker's host on the Internet side to which a valid user can connect to.

The DLD server detects the sensitive data within each packet on basis of a stateless filtering system. We define the sensitivity of a packet in Formula 2.

$$\mathcal{S}_{packet} = \frac{|\underset{p_d}{\gg}\ddot{\mathbb{S}}^* \cap \underset{p_d}{\gg}\tilde{\mathbb{T}}|}{min(|\mathbb{S}^*|, |\tilde{\mathbb{T}}|)} \times \frac{|\mathbb{S}^*|}{|\ddot{\mathbb{S}}^*|} \tag{2}$$

$\tilde{\mathbb{T}}$ is the set of all fingerprints extracted in a packet. $\mathbb{S}^*$ is the set of all sensitive fuzzy fingerprints. For each piece of sensitive data, data owner computes $\mathbb{S}^*$ and reveals a sample set $\ddot{\mathbb{S}}^*$ ($\ddot{\mathbb{S}}^* \subseteq \mathbb{S}^*$) to the DLD server. The operator $\underset{p_d}{\gg}$ indicates right shifting every fingerprint in a set by $p_d$ bits. The DLD server computes $\mathcal{S}_{packet}$ ($\mathcal{S}_{packet} \in [0,1]$) and compares it to a threshold $\mathcal{S}_{thres} \in (0,1)$. Packets with $\mathcal{S}_{packet} \geq \mathcal{S}_{thres}$ are marked sensitive.

Without the fuzzification phase, Formula 2 can be simplified to Formula 3. $\mathbb{S}$ is the set of all sensitive fingerprints, and $\ddot{\mathbb{S}}$ is the revealed fingerprints set.

$$\mathcal{S}_{packet} = \frac{|\ddot{\mathbb{S}} \cap \tilde{\mathbb{T}}|}{min(|\mathbb{S}|, |\tilde{\mathbb{T}}|)} \times \frac{|\mathbb{S}|}{|\ddot{\mathbb{S}}|} \tag{3}$$

Our current evaluation results reported are based on the simplified leak detection without the fuzzification phase. Additional experiments assessing the impact of fuzzification on privacy can be found in [28].

The goal of our evaluation is to answer the following questions:

1. Can our solution accurately detect sensitive data-leak in the traffic with low false positives (false alarms) and high true positives (real leaks)?
2. Does using partial sensitive-data fingerprints reduce the detection accuracy in our system?
3. What is the performance advantage of our *fingerprint filter* over traditional Bloom filter equipped with SHA-1?
4. How to choose a proper fuzzy length and make a balance between the privacy need and the number of alerts?
5. Can we experimentally validate the *min-wise independence* property of Rabin fingerprint?

The questions are experimentally addressed and answered in our following sections with the last two answered in our technical report [28].

## 5.1 Accuracy Evaluation

We generate 20,000 personal financial records as the sensitive data and store them in a text file. The data contains (fictitious) *person name*, *social security number*, *credit card number*, *credit card expiration date*, and *credit card CVV*.

To evaluate the accuracy of our strategy, we perform three separate experiments using the same sensitive dataset:

Exp.1 A user leaks the entire set of sensitive data via FTP by uploading it to a FTP server on the Internet.

Exp.2 (Base Line) The outbound HTTP traffic of Internet-surfing by 20 users are captured (30 minutes per user), and given to the DLD server to analyze, as a base line. No sensitive data (i.e., zero true positive) should be confirmed.

Exp.3 (Base Line) The Enron dataset (2.6 GB data, 150 users' 517,424 emails) as a virtual network traffic is given to the DLD server to analyze. Each virtual network packet created is based on an email in the dataset. No sensitive data (i.e., zero true positive) should be confirmed by the data onwer.

All sensitive fingerprints ($\mathbb{F}^D_{sens} = \mathbb{F}^A_{sens}$) are used in the detection, and the results are shown in Table 1. The first experiment is designed to infer the true positive rate. We manually check each packet and find out that the DLD server detects *all* 651 real sensitive packets (all of them have sensitivity values greater than 0.9). The sensitivity value is less than one, because the layered headers (IP, TCP, HTTP, etc.) in a packet are not sensitive. The next two experiments are designed to estimate the false positive rate. We found that none of the packets has a sensitivity value greater than 0.05, and the average sensitivity is very low. The results indicate that the algorithm performs as expected on plaintext.

| Dataset | Exp.1 | Exp.2 | Exp.3 |
|---|---|---|---|
| $\mathcal{S}_{packet}$ Mean | 0.952564 | 0.000005 | 0.001849 |
| $\mathcal{S}_{packet}$ STD | 0.004011 | 0.000133 | 0.002178 |

**Table 1.** Mean and standard deviations of the sensitivity per packet in three separate experiments. For Exp.1, the higher sensitivity, the better; for the other two (negative control), the lower sensitivity, the better.

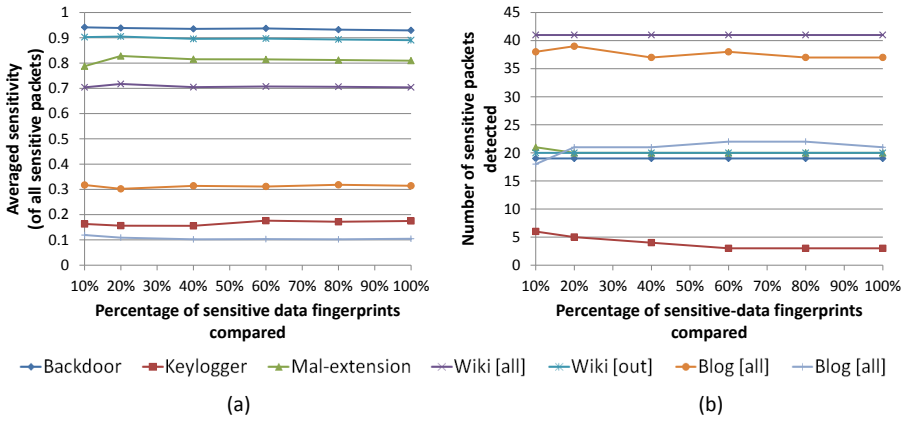


**Fig. 2.** Performance comparison in terms of (a) the averaged sensitivity and (b) the number of detected sensitive packets. X-axis, $\frac{|\mathbb{F}^D_{sens}|}{|\mathbb{F}^A_{sens}|}$, indicates the percentage of sensitive-data fingerprints revealed to the DLD server and used in the detection. [out] indicates outbound traffic only, while [all] means both outbound and inbound traffic captured and analyzed.

The data owner may reveal a subset of sensitive data's fingerprints to the DLD server for detection, as opposed to the entire set. We are particularly interested in measuring the percentage of revealed fingerprints that can be detected in the traffic, assuming that fingerprints are equally likely to be leaked (Given the *subset independence* property, sensitive-data's fingerprints are equally likely

to be selected for detection). We reproduce several real-world scenarios where data leaks are caused by human users or software applications.

- In the web-leak scenarios, a user posts sensitive data on wiki (MediaWiki) and blog(WordPress) pages.
- In the backdoor scenario, a program (*Glacier*) on the user's machine (Windows 7) leaks sensitive data.
- In the email-leak scenario, a malicious Firefox extension *FFsniFF* records the information in sensitive web forms, and emails the data to the attacker.
- In the keylogging scenario, a keylogger *EZRecKb* exports intercepted keystroke values on a user's host. The keylogger records every key stroke, replacing the function keys with labels, such as "[left shift]" in its log. *EZRecKb* connects to a pre-defined SMTP server on the Internet and sends its log periodically. In this experiment, the user manually type the text, simulating typos and corrections, which bring in modifications of the original sensitive data.

In these experiments, the source file of TCP/IP page on wikipedia (24KB in text) is used as the sensitive data. Partial fingerprints are revealed for detection, the sensitivity threshold is set $\mathcal{S}_{thres} = 0.05$, and plain set intersection test is used in DETECT operation.

Figure 2 shows the comparison of performance across various size of fingerprints used in the detection, in terms of the averaged sensitivity per packet in (a) and the number of detected sensitive packets in (b). These accuracy values reflect results computed by the data owner after running the POSTPROCESS operation. The results show that the use of partial sensitive-data fingerprints does not much degrade the detection rate compared to the use of full sets of sensitive-data fingerprints.

In Figure 2 (a), the sensitivities of experiments vary due to different levels of modification by the leaking programs, which makes it difficult to detect. WordPress converts space into "+" when sending the HTTP POST request. Keylogger inserts function-key as labels into the original text as well as typing typos and corrections. In Figure 2 (b), [all] results contain both outbound and inbound traffic and double the real number of sensitive packets in Blog and Wiki scenarios due to HTML fetching of the submitted data.

## 5.2 Runtime Comparison

Our fingerprint filter implementation is based on the Bloom filter library in Python (`Pybloom`). We compare the runtime of Bloom filter with SHA-1 and that of fingerprint filter with Rabin fingerprint. For Bloom filters and fingerprint filters, we test their performance with 2, 6, and 10 hash functions. We inspect 100 packets with random content against 10 pieces sensitive data of various length for each point drawn in Figure 3 – there are a total of 1,625,600 fingerprints generated from the traffic and 76,160 pieces of fingerprints from the sensitive data. We show the detection time per packet in Figure 3. The time used to create the filters during the sensitive data initialization is similar to the detection phase. Therefore it is not shown in the paper due to limited space.

The result indicates that fingerprint filters run faster than Bloom filters, which is expected as Rabin fingerprint is easier to compute than SHA-1. The gap is not significant due to the fact that Python uses a virtualization architecture. We have the core hash computations implemented in Python C/C++ extension, but the remaining control flow and function call statements are in pure Python. The performance difference between Rabin fingerprint and SHA-1 is large masked by the runtime overhead spent on non-hash related operations.

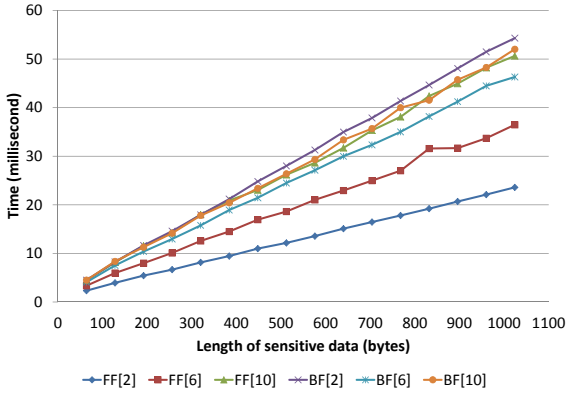


**Fig. 3.** The overhead of using the filters to detect data-leak, the runtime is per packet (averaged from 100 packets) against all 10 pieces of sensitive data, and the X-axis indicates the amount of sensitive information in a packet.

Using fewer hash functions in Bloom filters or fewer polynomials in the fingerprint filters produces more false positives at the DLD provider. The data owner can quickly identify real data leaks from reported leaked instances. This increased collision improves the data privacy. For example, Bloom filter with 10 hashes has a collision (false positive) probability of 0.10%, 6 hashes 1.56%, and 2 hashes 25%. We expect fingerprint filter to provide similar detection accuracy to the plain set intersection realization as reported in Section 5.1. Our fuzzy fingerprint should not be confused with fuzzy Bloom filter [22].

*Summary* Our detection rates in terms of the number of sensitive packets found do not decrease much with the decreasing size of fingerprint sets in Figure 2, even when only 10% of the sensitive-data fingerprints are used for detection. It is desirable for both privacy and efficiency considerations to have the data owner reveal as few fingerprints as possible. Our experiments evaluate several noisy conditions such as *data insertion* – for MediaWiki-based leak scenario, traffic contains extra HTML tags in addition to sensitive data, *data deletion* – traffic contains truncated sensitive data (not shown due to space limit), and *data substitution* – for the keylogger and WordPress-based leak scenarios, certain original data elements are replaced in the traffic. Our results indicate that the shingle-and-fingerprint method indeed can tolerate these three types of noises in the traffic to some degree. Our algorithm works well especially in the case

where consecutive data blocks are preserved (i.e., *local* data features are preserved) as in the MediaWiki-based leak scenario. When the noises spread across the data and destroy the local features (e.g., replacing every space with another character), the detection rate decreases as expected. The use of shorter shingles mitigates the problem, but may increase false positives. How to improve the noise tolerance property in those conditions remains an open problem. Our fuzzy fingerprint mechanism supports the detection of data-leak at various sizes and granularities. Our evaluation reported is run at the packet level. More fine-grained segment inspection may be needed for detecting smaller pieces of sensitive data leaked.

# 6 Related Work

Our fuzzy fingerprint method and its privacy-preserving feature enable its adopter to provide the data-leak detection as a service. Therefore, our technique distinguishes itself from existing commercial products (e.g., Global Velocity).

There have been several advances in developing *privacy-aware* collaborative solutions from both system [9, 21, 27] and theory perspectives [20, 34]. Specifically, Rabin fingerprint [24] based on shingles was used previously for identifying similar spam messages in a collaborative setting [21], as well as collaborative worm containment [9], virus scan [14], and fragment detection [25].

Our work fundamentally differs from the above shingle-based studies [9, 14] in particular. We consider the new problem of data-leak detection in a unique outsourced setting where the DLD provider is not fully trusted. Such privacy requirement does not exist in the virus-scan paradigm [14], for the virus signatures are non-sensitive. In comparison, data-leak detection is more challenging because of the additional privacy requirement, which limits the amount of data that can be used during the detection and the amount of sensitive information gained by the DLD provider. In the meantime, the provider's detection accuracy cannot be compromised with partial digests based on the sensitive data. Our fuzzy fingerprint method is new, and our work describes the first systematic solution to privacy-preserving data-leak detection with convincing results.

Information leak through outbound web traffic was studied by Borders and Prakash [3]. Both theirs and our work detect suspicious data flow on unencrypted network traffic. Their approach is based on the key observation that network traffic has high regularities and that information (e.g., header data) may be repeated. They proposed an elegant solution that detects any substantial increase in the amount of new information in the traffic. Their anomaly-detection method detects deviations from normal data-flow scenarios, which are captured in rules. In comparison, our work inspects traffic for signatures of sensitive-data and does not require any assumption on the patterns of normal header fields or payload. Furthermore, our solution provides privacy protection of the sensitive data against semi-honest DLD providers. We also give performance evidences indicating the efficiency of our solution in practice.

A black-box approach for data leak detection was proposed in [11], which expands local data tracking to a network-wide environment. It mainly focuses on

data confinement and detecting unauthorized sensitive data flow among forked processes. The specific goal makes it different from our approach to detect general data leaks over a network.

In the grid computing environment, the verification of outsourced execution was studied by Du and Goodrich in [12]. The method inserts chaff into input before outsourcing a job and verifies whether the chaff is processed or not after harvest. The threat models and security goals in our outsourced data-leak detection work and in [12] are fundamentally different.

The method of deep packet inspection is also widely used in network intrusion detection system (NIDS), such as SNORT [26] and Bro. They focus on designing and implementing efficient string matching algorithms [1] to handle short and flexible patterns in network traffic. However, NIDS is not designed for various kinds of sensitive data (e.g. long non-duplicated data), it may cause problems (e.g. large amount of states in an automaton) in data leak detection scenarios. On the contrary, our solution is not limited to very special types of sensitive data, and we provide an unique privacy-preserving feature for service outsourcing.

Encrypted traffic, which cannot be directly inspected [30], requires host-based DLD solutions to complement our network-based method. One approach is to instrument the kernel so that the inspection can be performed in the operating system of a host before data is encrypted. Existing approaches involving data flow and taint analysis [37] can be integrated.

An alternative to our approach for privacy-preserving computation is to use cryptographic mechanisms. Secure multi-party computation (SMC) is a research direction pioneered by Yao [35], where participants only learn the outcomes of computation, not the private inputs. Existing SMC solutions can support a wide range of fundamental arithmetic, set, and string operations as well as complex functions such as knapsack computation [36], automated trouble-shooting [15], network event statistics [8], private information retrieval [32], genomic computation [17], private join operations [10], and distributed data mining [16]. The provable privacy guarantees offered by SMC come at a cost in terms of computational complexity and implementation complexity as well. The advantage of our shingle/fingerprint based approach is much more efficient and simpler.

# 7 Conclusions and Future Work

We proposed a novel privacy-preserving data-leak detection model and its fuzzy fingerprint realization. Using special digests, the exposure of the sensitive data is kept to a minimum during the detection. We have conducted extensive experiments to validate the accuracy, privacy, and efficiency of our solutions. For future work, we plan to focus on designing a host-assisted mechanism for the complete data-leak detection for large-scale organizations.

# References

1. Aho, A. V., and Corasick, M. J. Efficient string matching: an aid to bibliographic search. *Commun. ACM* (1975).

2. Bohman, T., Cooper, C., and Frieze, A. M. Min-wise independent linear permutations. *Electr. J. Comb. 7* (2000).
3. Borders, K., and Prakash, A. Quantifying information leaks in outbound web traffic. In *Proceedings of the IEEE Symposium on Security and Privacy* (May 2009).
4. Borders, K., Weele, E. V., Lau, B., and Prakash, A. Protecting confidential data on personal computers with storage capsules. In *USENIX Security Symposium* (2009), USENIX Association, pp. 367–382.
5. Broder, A. Some applications of Rabins fingerprinting method. *Sequences II: Methods in Communications, Security, and Computer Science* (1993), 143–152.
6. Broder, A. Z. Identifying and filtering near-duplicate documents. In *COM '00: Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching* (London, UK, 2000), Springer-Verlag, pp. 1–10.
7. Broder, A. Z., Charikar, M., Frieze, A. M., and Mitzenmacher, M. Min-wise independent permutations. *Journal of Computer and System Sciences 60* (2000), 630 – 659.
8. Burkhart, M., Strasser, M., Many, D., and Dimitropoulos, X. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *Proceedings of USENIX Security* (2010).
9. Cai, M., Hwang, K., Kwok, Y.-K., Song, S., and Chen, Y. Collaborative Internet worm containment. *IEEE Security and Privacy 3*, 3 (2005), 25–33.
10. Carbunar, B., and Sion, R. Joining privately on outsourced data. In *Secure Data Management* (2010), W. Jonker and M. Petkovic, Eds., vol. 6358 of *Lecture Notes in Computer Science*, Springer, pp. 70–86.
11. Croft, J., and Caesar, M. Towards practical avoidance of information leakage in enterprise networks. In *USENIX HotSec*, August 2011.
12. Du, W., and Caesar, M. T. Searching for high-value rare events with uncheatable grid computing. In *Proceedings of the 5th International Conference on Applied Cryptography and Network Security (ACNS)* (2005).
13. Fawcett, T. W. EXFILD: A tool for the detection of data exfiltration using entropy and encryption characteristics of network traffic. Thesis submitted to Delaware University.
14. Hao, F., Kodialam, M., Lakshman, T. V., and Zhang, H. Fast payload-based flow estimation for traffic monitoring and network security. In *ANCS '05: Proceedings of the 2005 ACM symposium on Architecture for networking and communications systems* (New York, NY, USA, 2005), ACM, pp. 211–220.
15. Huang, Q., Jao, D., and Wang, H. J. Applications of secure electronic voting to automated privacy-preserving troubleshooting. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS)* (2005).
16. Jagannathan, G., and Wright, R. N. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining* (2005).
17. Jha, S., Kruger, L., and Shmatikov, V. Towards practical privacy for genomic computation. In *IEEE Symposium on Security and Privacy* (2008), IEEE Computer Society.,pp. 216–230.
18. Jiang, X., Wang, X., and Xu, D. Stealthy malware detection and monitoring through VMM-based "out-of-the-box" semantic view reconstruction. *ACM Trans. Inf. Syst. Secur. 13*, 2 (2010).
19. Jung, J., Sheth, A., Greenstein, B., Wetherall, D., Maganis, G., and Kohno, T. Privacy Oracle: a system for finding application leaks with black box differential testing. In *Proceedings of Computer and Communications Security (CCS)* (2008).

20. Kleinberg, J., Papadimitriou, C. H., and Raghavan, P. On the value of private information. In *TARK '01: Proceedings of the 8th conference on Theoretical aspects of rationality and knowledge* (San Francisco, CA, USA, 2001), Morgan Kaufmann Publishers Inc., pp. 249–257.
21. Li, K., Zhong, Z., and Ramaswamy, L. Privacy-aware collaborative spam filtering. *IEEE Transactions on Parallel and Distributed systems 20*, 5 (May 2009).
22. Mayer, C. P. Bloom filters and overlays for routing in pocket switched networks. In *Proceedings of ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT) Student Workshop* (2009).
23. Rabin, M. O. Digitalized signatures as intractable as factorization. Tech. Rep. MIT/LCS/TR-212, MIT Laboratory for Computer Science, Jan. 1979.
24. Rabin, M. O. Fingerprinting by random polynomials. Tech. rep., Center for Research in Computing Technology, Harvard University, 1981. TR-15-81.
25. Ramaswamy, L., Iyengar, A., Liu, L., and Douglis, F. Automatic detection of fragments in dynamically generated web pages. In *Proceedings of the 13th International World Wide Web Conference (WWW)* (May 2004).
26. Roesch, M. Snort: lightweight intrusion detection for networks In *Proceedings of the 13th Conference on Systems Administration (LISA-99)* (1999).
27. Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web* (2001).
28. Shu, X. and Yao, D. Data leak detection as a service: challenges and solutions. Technical Report TR-12-10, Computer Science, Virginia Tech (2012).
29. Stefan, D., Wu, C., Yao, D., and Xu, G. Cryptographic provenance verification for the integrity of keystrokes and outbound network traffic. In *Proceedings of the 8th International Conference on Applied Cryptography and Network Security (ACNS)* (June 2010).
30. Varadharajan, V. Internet filtering issues and challenges. *Journal of IEEE Security & Privacy* (2010), 62 – 65.
31. Wang, Y.-M., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen, S., and King, S. Automated web patrol with Strider HoneyMonkeys: Finding web sites that exploit browser vulnerabilities. In *Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS)* (2006).
32. Yoshida, R., Cui, Y., Sekino, T., Shigetomi, R., Otsuka, A., and Imai, H. Practical searching over encrypted data by private information retrieval. In *Proceedings of the Global Communications Conference (GLOBECOM)* (2010).
33. Xu, K., Yao, D., Ma, Q., and Crowell, A. Detecting infection onset with behavior-based policies. In *Proceedings of the Fifth International Conference on Network and System Security (NSS)* (September 2011).
34. Xu, S. Collaborative attack vs. collaborative defense. In *The 4th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)* (2008).
35. Yao, A. C. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science* (1986), IEEE Computer Society Press. , pp. 162–167.
36. Yao, D., Frikken, K. B., Atallah, M. J., and Tamassia, R. Private information: to reveal or not to reveal. *ACM Trans. Inf. Syst. Secur. 12*, 1 (2008).
37. Yin, H., Song, D., Egele, M., Kruegel, C., and Kirda, E. Panorama: Capturing system-wide information flow for malware detection and analysis. *In Proceedings of the 14th ACM Conferences on Computer and Communication Security (CCS)* (2007).