This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2015.2398363, IEEE Transactions on Information Forensics and Security

Privacy-Preserving Detection of Sensitive Data Exposure

Xiaokui Shu, Danfeng (Daphne) Yao, *Member, IEEE*, and Elisa Bertino, *Fellow, IEEE*

***Abstract*—Statistics from security firms, research institutions and government organizations show that the number of data-leak instances have grown rapidly in recent years. Among various data-leak cases, human mistakes are one of the main causes of data loss. There exist solutions detecting inadvertent sensitive data leaks caused by human mistakes and to provide alerts for organizations. A common approach is to screen content in storage and transmission for exposed sensitive information. Such an approach usually requires the detection operation to be conducted in secrecy. However, this secrecy requirement is challenging to satisfy in practice, as detection servers may be compromised or outsourced. In this paper, we present a privacy-preserving data-leak detection solution to solve the issue where a special set of sensitive data digests is used in detection. The advantage of our method is that it enables the data owner to safely delegate the detection operation to a semi-honest provider without revealing the sensitive data to the provider. We describe how ISPs can offer their customers data-leak detection as an add-on service with strong privacy guarantees. The evaluation results show that our method can support accurate detection with very small number of false alarms under various data-leak scenarios.**

***Index Terms*—Data leak, network security, privacy, collection intersection**

I. Introduction

ACCORDING to a report from Risk Based Security (RBS) [2], the number of leaked sensitive data records has increased dramatically during the last few years, i.e., from 412 million in 2012 to 822 million in 2013. Deliberately planned attacks, inadvertent leaks (e.g., forwarding confidential emails to unclassified email accounts), and human mistakes (e.g., assigning the wrong privilege) lead to most of the data-leak incidents [3].

Detecting and preventing data leaks requires a set of complementary solutions, which may include data-leak detection [4], [5], data confinement [6], [7], [8], stealthy malware detection [9], [10], and policy enforcement [11].

Network data-leak detection (DLD) typically performs deep packet inspection (DPI) and searches for any occurrences of sensitive data patterns. DPI is a technique to analyze payloads of IP/TCP packets for inspecting application layer data, e.g., HTTP header/content. Alerts are triggered when the amount of sensitive data found in traffic passes a threshold. The detection system can be deployed on a router or integrated into existing network intrusion detection systems (NIDS).

Straightforward realizations of data-leak detection require the plaintext sensitive data. However, this requirement is undesirable, as it may threaten the confidentiality of the sensitive information. If a detection system is compromised, then it may expose the plaintext sensitive data (in memory). In addition, the data owner may need to outsource the data-leak detection to providers, but may be unwilling to reveal the plaintext sensitive data to them. Therefore, one needs new data-leak detection solutions that allow the providers to scan content for leaks without learning the sensitive information.

In this paper, we propose a data-leak detection solution which can be outsourced and be deployed in a semi-honest detection environment. We design, implement, and evaluate our *fuzzy fingerprint* technique that enhances data privacy during data-leak detection operations. Our approach is based on a fast and practical one-way computation on the sensitive data (SSN records, classified documents, sensitive emails, etc.). It enables the data owner to securely delegate the content-inspection task to DLD providers without exposing the sensitive data. Using our detection method, the DLD provider, who is modeled as an honest-but-curious (aka semi-honest) adversary, can only gain limited knowledge about the sensitive data from either the released digests, or the content being inspected. Using our techniques, an Internet service provider (ISP) can perform detection on its customers' traffic securely and provide data-leak detection as an add-on service for its customers. In another scenario, individuals can mark their own sensitive data and ask the administrator of their local network to detect data leaks for them.

In our detection procedure, the data owner computes a special set of digests or fingerprints from the sensitive data and then discloses only a small amount of them to the DLD provider. The DLD provider computes fingerprints from network traffic and identifies potential leaks in them. To prevent the DLD provider from gathering exact knowledge about the sensitive data, the collection of potential leaks is composed of real leaks and noises. It is the data owner, who post-processes the potential leaks sent back by the DLD provider and determines whether there is any real data leak.

In this paper, we present details of our solution and provide extensive experimental evidences and theoretical analyses to demonstrate the feasibility and effectiveness of our approach.

This work has been supported in part by Security and Software Engineering Research Center (S^2ERC), a NSF sponsored multi-university Industry/University Cooperative Research Center (I/UCRC).

A preliminary version of the work appeared in the Proceedings of the 8th International Conference on Security and Privacy in Communication Networks (SecureComm '12) [1].

X. Shu and D. Yao are with the Department of Computer Science, Virginia Tech, Blacksburg, VA, 24060 USA e-mail: {subx, danfeng}@vt.edu

E. Bertino is with the Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA e-mail: bertino@cs.purdue.edu

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2015.2398363, IEEE Transactions on Information Forensics and Security

Our contributions are summarized as follows.

1) We describe a privacy-preserving data-leak detection model for preventing inadvertent data leak in network traffic. Our model supports detection operation delegation and ISPs can provide data-leak detection as an add-on service to their customers using our model.
   We design, implement, and evaluate an efficient technique, fuzzy fingerprint, for privacy-preserving data-leak detection. Fuzzy fingerprints are special sensitive data digests prepared by the data owner for release to 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 its privacy guarantee, detection rate and efficiency. Our results indicate high accuracy achieved by our underlying scheme with very low false positive rate. Our results also show that the detection accuracy does not degrade much when only partial (sampled) sensitive-data digests are used. In addition, we give an empirical analysis of our fuzzification as well as of the fairness of fingerprint partial disclosure.

II. Model and Overview

We abstract the privacy-preserving data-leak detection problem with a threat model, a security goal and a privacy goal. First we describe the two most important players in our abstract 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 II-A and Section II-B.

A. 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 supervised network channels. Inadvertent data leak may be due to human errors such as forgetting to use encryption, carelessly forwarding an internal email and attachments to outsiders, or due to application flaws (such as described in [12]). A supervised network channel could be an unencrypted channel or an encrypted channel where the content in it can be extracted and checked by an authority. Such a channel is widely used for advanced NIDS where MITM (man-in-the-middle) SSL sessions are established instead of normal SSL sessions [13].
- Case II *Malicious data leak*: A rogue insider or a piece of stealthy software may steal sensitive personal or organizational data from a host. Because the malicious adversary can use strong private encryption, steganography or covert channels to disable content-based traffic inspection, this type of leaks is out of the scope of our network-based solution. Host-based defenses (such as detecting the infection onset [14]) 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 the data owner is aware of legitimate data transfers and permits such transfers. So the data owner can tell whether a piece of sensitive data in the network traffic is a leak using legitimate data transfer policies.

The security goal in this paper is to detect Case I leaks, that is inadvertent data leaks over supervised network channels. In other words, we aim to discover sensitive data appearance in network traffic over supervised network channels. We assume that: *i)* plaintext data in supervised network channels can be extracted for inspection; *ii)* the data owner is aware of legitimate data transfers (Case III); and *iii)* whenever sensitive data is found in network traffic, the data owner can decide whether or not it is a data leak. Network-based security approaches are ineffective against data leaks caused by malware or rogue insiders as in Case II, because the intruder may use strong encryption when transmitting the data, and both the encryption algorithm and the key could be unknown to the DLD provider.

B. Privacy Goal and Threat Model

To prevent the DLD provider from gaining knowledge of 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 a probability greater than $\frac{1}{K}$, 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 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 of such an approach is much higher than $\frac{1}{K}$ when there is no leak in traffic, because the adversary's inference can only be gained through brute-force guesses.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2015.2398363, IEEE Transactions on Information Forensics and Security

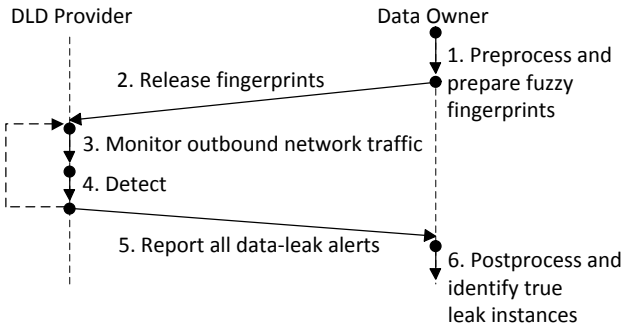


Fig. 1: Our Privacy-preserving Data-Leak Detection Model

The traffic content is accessible by the DLD provider in plaintext. Therefore, in the event of a data leak, the DLD provider may learn sensitive information from the traffic, which is inevitable for all deep packet inspection approaches. Our solution confines the amount of maximal information learned during the detection and provides quantitative guarantee for data privacy.

C. 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. Fig. 1 lists the six operations executed by the data owner and the DLD provider in our protocol. They 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. Details are presented in the next section.

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 a 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 representatives of local data patterns, e.g., when byte b_2 appears in the sensitive data, it is usually surrounded by bytes b_1 and b_3 forming a local pattern b_1, b_2, b_3. Local features preserve data patterns even when modifications (insertion, deletion, and substitution) are made to parts of the data. For example, if a byte b_4 is inserted after b_3, the local pattern b_1, b_2, b_3 is retained though the global pattern (e.g., a hash of the entire document) is destroyed. To achieve the privacy goal, 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 a crowd. It prevents the DLD provider from learning its exact value. We describe the technical details next.

III. FUZZY FINGERPRINT METHOD AND PROTOCOL

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

A. Shingles and Fingerprints

The DLD provider obtains digests of sensitive data from the data owner. The data owner uses a sliding window and Rabin fingerprint algorithm [15] to generate short and hard-to-reverse (i.e., one-way) digests through the fast polynomial modulus operation. The sliding window generates small fragments of the processed data (sensitive data or network traffic), which preserves the local features of the data and provides the noise tolerance property. Rabin fingerprints are computed as polynomial modulus operations, and can be implemented with fast XOR, shift, and table look-up operations. The Rabin fingerprint algorithm has a unique min-wise independence property [16], which supports fast random fingerprints selection (in uniform distribution) for partial fingerprints disclosure.

The shingle-and-fingerprint process is defined as follows. A sliding window is used to generate q-grams on an input binary string first. The fingerprints of q-grams are then computed.

A shingle (q-gram) is a fixed-size sequence of contiguous bytes. For example, the 3-gram shingle set of string `abcdefgh` consists of six elements {`abc`, `bcd`, `cde`, `def`, `efg`, `fgh`}. Local feature preservation is accomplished through the use of shingles. Therefore, our approach can tolerate sensitive data modification to some extent, e.g., inserted tags, small amount of character substitution, and lightly reformatted data. The use of shingles for finding duplicate web documents first appeared in [17], [18].

The use of shingles alone does not satisfy the one-wayness requirement. Rabin fingerprint is utilized to satisfy such requirement after shingling. In fingerprinting, each shingle is treated as a polynomial $q(x)$. Each coefficient of $q(x)$, i.e., c_i $(0 < i < k)$, is one bit in the shingle. $q(x)$ is mod by a selected irreducible polynomial $p(x)$. The process shown in (1) maps a k-bit shingle into a p_f-bit fingerprint f where the degree of $p(x)$ is $p_f + 1$.

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

From the detection perspective, a straightforward method is for the DLD provider to raise an alert if any sensitive fingerprint matches the fingerprints from the traffic[1]. However, this approach has a privacy issue. If there is a data leak, 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.

[1]In reality, data-leak detection solutions usually utilize more complex statistical models to raise alerts instead of alerting individual fingerprints. Statistical approaches, e.g., packet sensitivity in Section V, eliminate accidental matches.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2015.2398363, IEEE Transactions on Information Forensics and Security

We propose an 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 define the notions of *fuzzy length* and *fuzzy set* next and then describe how they are used in our detailed protocol in Section III-B.

Definition 1. *Given a p_f-bit-long fingerprint f, the fuzzy length p_d ($p_d < p_f$) is the number of 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 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.

B. Operations in Our Protocol

1) PREPROCESS: This operation is run by the data owner on each piece of sensitive data.
   a) The data owner chooses four public parameters $(q, p(x), p_d, M)$. q is the length of a shingle. $p(x)$, is an irreducible polynomial (degree of p_f+1) used in Rabin fingerprint. Each fingerprint is p_f-bit long and the fuzzy length is p_d. M is a bitmask, which is p_f-bit long and contains p_d `0`'s at random positions. The positions of `1`'s and `0`'s in M indicate the bits to preserve and to randomize in the fuzzification, respectively.
   b) The data owner computes $\mathbb{S}$, which is the set of all Rabin fingerprints of the piece of sensitive data.
   c) The data owner transforms each fingerprint $f \in \mathbb{S}$ into a fuzzy fingerprint f^* with randomized bits (specified by the mask M). The procedure is described as follows: for each $f \in \mathbb{S}$, the data owner generates a random p_f-bit binary string $\dot{f}$, mask out the bits not randomized by $\dot{f}' = (\texttt{NOT}\ M)\ \texttt{AND}\ \dot{f}$ (`1`'s in M indicate positions of bits not to randomize), and fuzzify f with $f^* = f\ \texttt{XOR}\ \dot{f}'$. The overall computation is described in (2).

$$f^* = ((\texttt{NOT}\ M)\ \texttt{AND}\ \dot{f})\ \texttt{XOR}\ f \tag{2}$$

   All fuzzy fingerprints are collected and form the output of this operation, the fuzzy fingerprint set, $\mathbb{S}^*$.

2) RELEASE: This operation is run by the data owner. The fuzzy fingerprint set $\mathbb{S}^*$ obtained by PREPROCESS is released to the DLD provider for use in the detection, along with the public parameters $(q, p(x), p_d, M)$. The data owner keeps $\mathbb{S}$ for use in the subsequent POSTPROCESS operation.

3) MONITOR: This operation is run by the DLD provider. The DLD provider monitors the network traffic T from the data owner's organization. Each packet in T is collected and the payload of it is sent to the next operation as the network traffic (binary) string $\tilde{T}$.
   The payload of each packet is not the only choice to define $\tilde{T}$. A more sophisticated approach could identify TCP flows and extract contents in a TCP session as $\tilde{T}$. Contents of other protocols can also be retrieved if required by the detection metrics.

4) DETECT: This operation is run by the DLD provider on each $\tilde{T}$ as follows.
   a) The DLD provider first computes the set of Rabin fingerprints of traffic content $\tilde{T}$ based on the public parameters. The set is denoted as $\mathbb{T}$.
   b) The DLD provider tests whether each fingerprint $f' \in \mathbb{T}$ is also in $\mathbb{S}^*$ using the fuzzy equivalence test (3).

$$E(f', f^*) = \texttt{NOT}\ (M\ \texttt{AND}\ (f'\ \texttt{XOR}\ f^*)) \tag{3}$$

   $E(f', f^*)$ is either `True` or `False`. $f'\ \texttt{XOR}\ f^*$ gives the difference between f' and f^*. $M\ \texttt{AND}\ (f'\ \texttt{XOR}\ f^*)$ filters the result leaving only the interesting bits (preserved bits with `1`'s in M). Because `XOR` yields `0` for equivalent bits, `NOT` is used to turn `0`-bits into `1`'s (and `1`'s into `0`'s). The overall result from (3) is read as a boolean indicating whether or not f' is equivalent to a fuzzy fingerprint $f^* \in \mathbb{S}^*$.
   (2) and (3) are designed in a pair, and M works the same in both equations by masking out fuzzified bits at same positions in each f, f^* and f'. All f' with `True` values are record in a set $\hat{\mathbb{T}}$.
   c) The DLD provider aggregates the outputs from the preceding step and raises alerts based on a threshold. Our concrete aggregation formula is given in section V.

5) REPORT: If DETECTION on $\tilde{T}$ yields an alert, the DLD provider reports the set of detected candidate leak instances $\hat{\mathbb{T}}$ to the data owner.

6) POSTPROCESS: After receiving $\hat{\mathbb{T}}$, the data owner test every $f' \in \hat{\mathbb{T}}$ to see whether it is in $\mathbb{S}$. A precise likelihood of data leaking is computed at the data owner's, which we discuss more in section V.

In the protocol, because S_{f^*,p_d}, the fuzzy set of f^*, includes the original fingerprint f, the true data leak can be detected (i.e., true positive). Yet, due to the increased detection range, multiple values in S_{f^*,p_d} may trigger alerts. Because S_{f^*,p_d} 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 IV.

The DETECT operation can be performed between $\mathbb{T}$ and $\mathbb{S}^*$ via set intersection test with a special equivalence test function (e.g. Formula 5 in Section V 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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2015.2398363, IEEE Transactions on Information Forensics and Security

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. There exists an obvious tradeoff between privacy and detection efficiency – large fuzzy set allows a fingerprint to hide among others and confuses the DLD provider, yet this indistinguishability results in more work in POSTPROCESS. We provide quantitative analysis on fuzzy fingerprint including empirical results on different sizes of fuzzy sets.

C. Extensions

Fingerprint Filter We develop this extension to use Bloom filter in the DETECT operation for efficient set intersection test. Bloom filter [19] is a well-known space-saving data structure for performing set-membership test. It applies multiple hash functions to each of the set elements and stores the resulting values in a bit vector; to test whether a value v belongs to the set, the filter checks each corresponding bit mapped with each hash function. Bloom filter in combination with Rabin fingerprint is referred to by us as the *fingerprint filter*. We use Rabin fingerprints with variety of modulus's in fingerprint filter as the hash functions, and we perform extensive experimental evaluation on both fingerprint filter and bloom filter with MD5/SHA in Section V.

Partial disclosure Using the min-wise independent property of Rabin fingerprint, the data owner can quickly disclose partial fuzzy fingerprints to the DLD provider. The purpose of partial disclosure is two-fold: *i)* to increase the scalability of the comparison in the DETECT operation, and *ii)* to reduce the exposure of data to the DLD provider for privacy. The method of partial release of sensitive data fingerprints is similar to the suppression technique in database anonymization [20], [21].

This extension requires a good uniform distribution random selection to avoid disclosure bias. The min-wise independence feature of Rabin fingerprint guarantees that the minimal fingerprint is coming from a (uniformly distributed) random shingle. The property is also valid for a minimum set of fingerprints and so the data owner can just select r smallest elements in $\mathbb{S}^*$ to perform partial disclosure. The r elements are then sent to the DLD provider in RELEASE operation instead of $\mathbb{S}^*$. We implement the partial disclosure policy, evaluate its influence on detection rate, and verify the min-wise independence property of Rabin fingerprint in Section V.

IV. 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. We point out the limitations associated with the proposed network-based DLD approaches.

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 as follows.

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.

We explain our quantification in two scenarios:

i) There is a match between a sensitive fuzzy fingerprint f^* (derived from the sensitive fingerprint f) and fingerprints from the network traffic. Because the size of fuzzy set S_{f,p_d} is upper bounded by 2^{p_d} (Definition 2), there could be at most 2^{p_d} (sensitive or non-sensitive) fingerprints fuzzified into the identical f^*. Given a set (size n) of traffic fingerprints, the DLD provider expects to find K fingerprints matched to f^* where $K = \frac{n}{2^{p_f}} \times 2^{p_d}$.
   a) If f corresponds to a sensitive shingle leaked in the traffic, i.e., f is within the K traffic fingerprints, the likelihood of correctly pinpointing f from the K fingerprints is $\frac{1}{K}$, or $\frac{2^{p_f-p_d}}{n}$. The likelihood is fare because both sensitive data and network traffic contain binary data. It is difficult to predict the subspace of the sensitive data in the entire binary space.
   b) If the shingle form of f is not leaked in the traffic, the DLD provider cannot use the K traffic fingerprints, which match f^*, to infer f. Alternatively, the DLD provider needs to brute force f^* to get f, which is discussed as in the case of no match.

ii) There is no match between sensitive and traffic fingerprints. The adversarial DLD provider needs to brute force reverse the Rabin fingerprinting computation to obtain the sensitive shingle. There are two difficulties in reversing a fingerprint: *i)* Rabin fingerprint is a one-way hash. *ii)* Multiple shingles can map to the same fingerprint. It requires to searching the complete set of possible shingles for a fingerprint and to identify the sensitive one from the set. This brute-force attack is difficult for a polynomial-time adversary, thus the success probability is not included.

In summary, the DLD provider cannot decide whether the alerts (traffic fingerprints matched f^*) contain any leak or not (case i.a or i.b). Even if it is known that there is a real leak in the network traffic, the polynomial-time DLD provider has no greater than $\frac{2^{p_f-p_d}}{n}$ probability of correctly pinpointing a sensitive shingle (case i.a).

Alert Rate We quantify 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 τ, the expected traffic fingerprints set size n, fingerprint length p_f, fuzzy length p_d, partial disclosure rate $p_s \in (0, 1]$, and the expected rate α, which is the percentage of fingerprints in the sensitive

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2015.2398363, IEEE Transactions on Information Forensics and Security

data entry that appear in the network traffic. The expected alert rate R is presented in (4).

$$R = \frac{\alpha p_s K \tau}{n} = \frac{\alpha p_s \tau}{2^{p_f - p_d}} \tag{4}$$

R is used to derive threshold $\mathcal{S}_{thres}$ in the detection; $\mathcal{S}_{thres}$ should be lower than R. The overhead of our privacy-preserving approach over traditional fingerprinting data-leak detection solutions is tightly related to R and $\mathcal{S}_{thres}$, because there is an extra POSTPROCESS step in our approach after the DLD provider detects potential leaks. The less potential leaks the DLD provider reports back to the data owner, the less overhead is introduced by our privacy-preserving approach, while the less privacy is achieved since K is small.

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 [18], 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 [17] and are negligible.

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. A restricted space limits K and can expose the fuzzified fingerprint. For instance, one may assume that a password has higher entropy than normal English shingles, thus a fuzzy fingerprint of a password rules out much space of S_{f,p_d} where normal English lives. The full space with a variety of text encodings and binaries ensures that the major space is still there shadowing the fuzzy fingerprint.

Short polynomial modulus A naive alternative to the fuzzy fingerprint mechanism is to use a shorter polynomial modulus to compute Rabin fingerprints (e.g., 16-bit instead of 32-bit). This approach increases collisions for fuzzification purpose. However, one issue of this approach is that true positive and false positives yield the same fingerprint value due to collision, which prevents the data owner from telling true positives apart from false positives. In addition, our fuzzy fingerprint approach is more flexible from the deployment perspective, as the data owner can adjust and fine-tune the privacy and accuracy in the detection without recomputing the fingerprints. In contrast, the precision is fixed in the naive shorter polynomial approach unless fingerprints are recomputed.

Limitations: we point out three major limitations of our detection approach within our threat model.

Modified data leak The underlying shingle scheme of our approach has limited power to capture heavily modified data leaks. False negatives (i.e., failure to detect data leak) may occur due to the data modification (e.g., reformatting). 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. The modified data-leak detection problem is a general problem for all comparison-based data-leak detection solutions. More advanced content comparison techniques than shingles/fingerprints are needed to fully address the issue.

Dynamic sensitive 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.

Selective fragments leak The partial disclosure scheme may result in false negatives, i.e., the leaked data may evade the detection because it is not covered by the released fingerprints. This issue illustrates the tradeoff among detection accuracy, privacy guarantee and detection efficiency. Fortunately, it is expensive for an attacker to escape the detection with partial disclosure. On one hand, Rabin fingerprint guarantees that every fingerprint has the same probability to be selected and released through its min-wise independence property. Deliberately choosing unreleased segments from sensitive data is not easy. On the other hand, even figuring out which fingerprints are not released, one needs leaking inconsecutive bytes to bypass the detection. It usually makes no sense to leak inconsecutive bytes from sensitive data. Some format, e.g., binary, may be destroyed through the leaking.

V. EXPERIMENTAL EVALUATION

We implement our fuzzy fingerprint framework in Python, including packet collection, shingling, Rabin fingerprinting, as well as partial disclosure and fingerprint filter extensions. Our implementation of Rabin fingerprint is based on cyclic redundancy code (CRC). We use the padding scheme mentioned in [22] to handle small inputs. 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 networking environment in VirtualBox, and make a scenario where the sensitive data is leaked from a local network to the Internet. Multiple users' hosts (Windows 7) are put in the local network, which connect to the Internet via a gateway (Fedora). Multiple servers (HTTP, FTP, etc.) and an attacker-controlled host are put on the Internet side. 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. The speed aspect of privacy-preserving data-leak detection is another topic and we study it in [23].

In our prototype system, 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 (5), which is used by the DLD provider in DETECTION. It indicates the likelihood of a packet containing sensitive data.

$$\mathcal{S}_{packet} = \frac{|\gg_{p_d}^{\ddot{\mathbb{S}}^*} \cap \gg_{p_d}^{\mathbb{T}}|}{min(|\mathbb{S}^*|, |\mathbb{T}|)} \times \frac{|\mathbb{S}^*|}{|\ddot{\mathbb{S}}^*|} \tag{5}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2015.2398363, IEEE Transactions on Information Forensics and Security

$\mathbb{T}$ 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, the data owner computes $\mathbb{S}^*$ and reveals a partial set $\ddot{\mathbb{S}}^*$ ($\ddot{\mathbb{S}}^* \subseteq \mathbb{S}^*$) to the DLD provider. The operator $\gg_t$ indicates right shifting every fingerprint in a set by t bits, which is the implementation of a simple mask M in our protocol (Section III-B). $|\mathbb{S}^*|/|\ddot{\mathbb{S}}^*|$ estimates the leaking level of $\mathbb{S}^*$ according to the revealed and tested partial set $\ddot{\mathbb{S}}^*$. When too few fuzzy fingerprints are revealed, e.g., 10%, the samples may not sufficiently describe the leaking characteristic of the traffic, and the estimation can be imprecise. For each packet, the DLD server computes $\mathcal{S}_{packet}$ ($\mathcal{S}_{packet} \in [0, 1]$). If it is higher than a threshold $\mathcal{S}_{thres} \in (0, 1)$, $\mathbb{T}$ is reported back to the data owner, and the data owner uses (6) to determine whether it is a real leak in POSTPROCESS.

$$\mathcal{S}_{packet} = \frac{|\mathbb{S} \cap \mathbb{T}|}{min(|\mathbb{S}|, |\mathbb{T}|)} \quad (6)$$

The difference between (5) operated by the DLD provider and (6) by the data owner is that the original fingerprints $\mathbb{S}$ are used in (6) instead of the sampled and fuzzified set $\ddot{\mathbb{S}}^*$ in (5), so the data owner can pinpoint the exact leaks.

The use of $\mathcal{S}_{packet}$ and $\mathcal{S}_{thres}$ for detection is important because individual shingles or fingerprints are not accurate features to represent an entire piece of sensitive data. Sensitive data can share strings with non-sensitive data, e.g., formatting strings, which results in occasionally reported sensitive fingerprints. $\mathcal{S}_{packet}$ is an accumulated score and $\mathcal{S}_{thres}$ filters out packets with occasionally discovered sensitive fingerprints.

The evaluation goal 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 with SHA-1?
4) How to choose a proper fuzzy length and make a balance between the privacy need and the number of alerts?

In the following subsection, we experimentally addressed and answered all the questions. For the first three questions, we present results based on the $\mathcal{S}_{packet}$ value calculated in (6). The first and second questions are answered in Section V-A. The third question is discussed in Section V-B. The last question is designed to understand the properties of fuzzification and partial disclosure, and it is addressed in Section V-C.

A. Accuracy Evaluation

We evaluate the detection accuracy in simple and complex leaking scenarios. First we test the detection rate and false positive rate in three simple experiments where the sensitive data is leaked in its original form or not leaked. Then we present accuracy evaluation on more complex leaking experiments to reproduce various real-world leaking detection scenarios.

Simple leaking scenarios. We test our prototype without partial disclosure in simple leaking scenarios, i.e., $\ddot{\mathbb{S}}^* = \mathbb{S}^*$. We generate 20,000 personal financial records as the sensitive data and store them in a text file. The data contains *person name*, *social security number*, *credit card number*, *credit card expiration date*, and *credit card CVV*.

TABLE I: 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

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

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

Exp.1 *True leak* A user leaks the entire set of sensitive data via FTP by uploading it to a remote FTP server.

Exp.2 *No leak* The non-related outbound HTTP traffic of 20 users is captured (30 minutes per user), and given to the DLD server to analyze. No sensitive data (i.e., zero true positive) should be confirmed.

Exp.3 *No leak* 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 owner.

The detection results are shown in Table I. Among the three experiments, the first one is designed to infer true positive rate. We manually check each packet and 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 high-layer headers (e.g., HTTP) 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. The results indicate that our design performs as expected on plaintext.

Complex leaking scenarios. 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[2]. We reproduce four real-world scenarios where data leaks are caused by human users or software applications.

Exp.4 *Web leak:* a user posts sensitive data on wiki (`MediaWiki`) and blog (`WordPress`) pages.

Exp.5 *Backdoor leak:* a program (`Glacier`) on the user's machine (Windows 7) leaks sensitive data.

Exp.6 *Browser leak:* a malicious Firefox extension `FFsniFF` records the information in sensitive web forms, and emails the data to the attacker.

Exp.7 *Keylogging leak:* a keylogger `EZRecKb` exports intercepted keystroke values on a user's host[3]. It

[2]Given the *subset independence* property, sensitive-data's fingerprints are equally likely to be selected for detection.

[3]`EZRecKb` records every key stroke and replaces the function keys with labels, such as `[left shift]`.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2015.2398363, IEEE Transactions on Information Forensics and Security

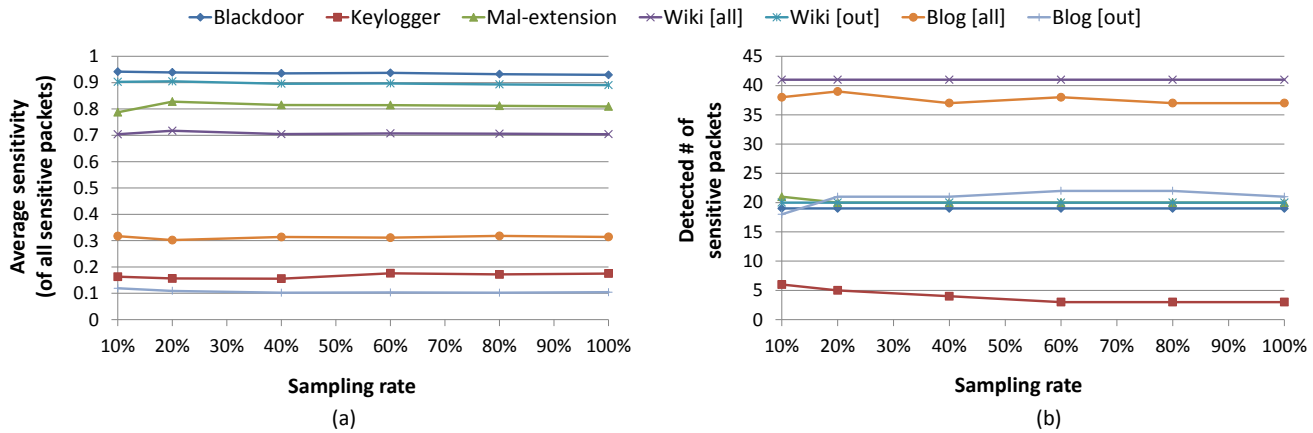


Fig. 2: Detection accuracy comparison in terms of (a) the averaged sensitivity and (b) the number of detected sensitive packets. X-axis is the partial disclosure rate, or the percentage of sensitive-data fingerprints revealed to the DLD server and used in the detection. [out] indicates outbound traffic only, while [all] means both outbound and inbound traffic captured and analyzed.

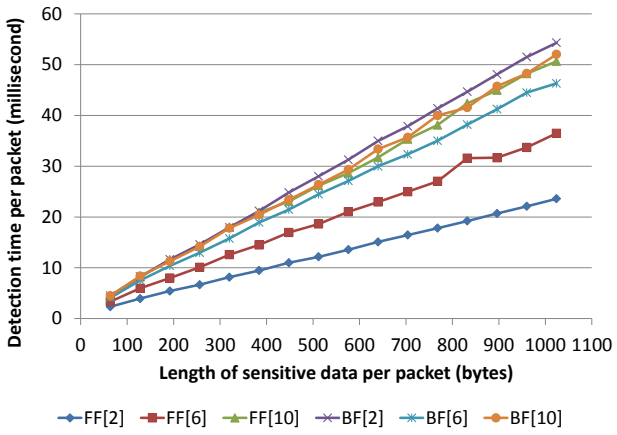
connects to a SMTP server on the Internet side and sends its log of keystrokes periodically.

In these four experiments, the source file of TCP/IP page on Wikipedia (24KB in text) is used as the sensitive data. The detection is performed at various partial disclosure rate. The subset of the sensitive fingerprints is selected randomly. The sensitivity threshold is $\mathcal{S}_{thres} = 0.05$.

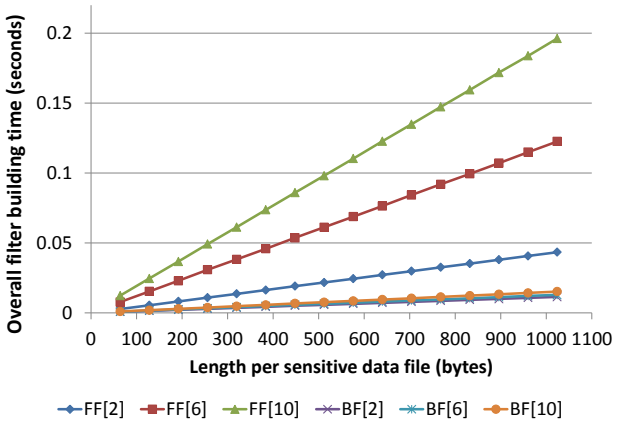
Fig. 2 shows the comparison of performance across various size of fingerprints used in the detection, in terms of the averaged sensitivity per packet in (a) and the number of detected sensitive packets in (b). These accuracy values reflect results of the POSTPROCESS operation.

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. However, extreme small sampling rates, e.g., 10%, may not provide sufficient numbers of fingerprints to describe the leaking characteristic of the traffic. The packet sensitivity estimation ($|\mathbb{S}|/|\hat{\mathbb{S}}|$ in (6)) magnifies the signal (the real sensitivity of a packet) as well as the noise produced by fingerprint sampling. The precision could be affected and drops, e.g., 6 packets with 10% vs. 3 packets with 100% for `Keylogger` in Fig. 2 (b).

In Fig. 2 (a), the sensitivities of experiments vary due to different levels of modification by the leaking programs, which makes it difficult to perform detection. `WordPress` substitutes spaces with +'s when sending the HTTP POST request. `EZRecKb` inserts function-key as labels into the original text. Typing typos and corrections also bring in modification to the original sensitive data. In Fig. 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 and displaying of the submitted data.



(a) Detection Time



(b) Filter Building Time

Fig. 3: The overhead of using the filters to detect data leaks. The detection time is averaged from 100 packets against all 10 pieces of sensitive data.

B. Runtime Comparison

Our fingerprint filter implementation is based on the Bloom filter library in Python (`Pybloom`). We compare the runtime of Bloom filter provided by standard `Pybloom` (with dynamically selected hash function from MD5, SHA-1, SHA-256,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2015.2398363, IEEE Transactions on Information Forensics and Security

SHA-384 and SHA-512) 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 at various lengths – there are a total of 1,625,600 fingerprints generated from the traffic and 76,160 pieces of fingerprints from the sensitive data.

We present the time for detection per packet in Fig. 3 (a). It shows that fingerprint filters run faster than Bloom filters, which is expected as Rabin fingerprint is easier to compute than MD5/SHA. 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 MD5/SHA is largely masked by the runtime overhead spent on non-hash related operations.

In Fig. 3 (a), the number of hash functions used in Bloom filters does not significantly impact their runtime, because only one hash function is operated in most cases for Bloom filters. `Pybloom` automatically chooses SHA-256 for Bloom filter with 6 hash functions and SHA-384 for Bloom filter with 10 hash functions. One hash is sufficient to distinguish 32-bits fingerprints. MD5 is automatically chosen for the Bloom filter with 2 hash functions, which gives more collisions and the second hash could be involved. We speculate this is the reason why Bloom filter with 2 hashes is slower than Bloom filters with 6 or 10 hashes. All of our fingerprint filters use 32-bit Rabin fingerprint functions. The small output space requires more than one hash for a membership test, so there is more significant overhead when a fingerprint filter is equipped with more hashes (6 vs. 2 and 10 vs. 6).

The filter construction time is shown in Fig. 3 (b). It shares similar characteristics with the detection time. Filters with more hash functions require more time to initialize, because every hash function need to be computed. The construction of fingerprint filters, especially assigning the irreducible polynomials $p(x)$ for each Rabin fingerprint, is written in pure Python, which is significantly slower than SHA-256 and SHA-384 encapsulated using Python C/C++ extension.

C. Sizes of Fuzzy Sets vs. Fuzzy Length

The size of fuzzy set corresponds to the K value in our definition of privacy goal. The higher K is, the more difficult it is for a DLD provider to infer the original sensitive data using our fuzzy fingerprinting mechanism – the fingerprint of the sensitive data hides among its neighboring fingerprints.

We empirically evaluate the average size of the fuzzy set associated with a given fuzzy length with both Brown Corpus (text) and real-world network traffic (text & binary).

- *Brown Corpus:* The Brown University Standard Corpus of Present-Day American English [24]. It contains 500 samples of English text across 15 genres, and there are 1,014,312 words in total.
- *Network traffic:* 500MB Internet traffic dump collected by us on a single host. It includes a variety of network traffic: multimedia Internet surfing (images, video, etc.), binary downloading, software and system updates, user profile synchronization, etc.

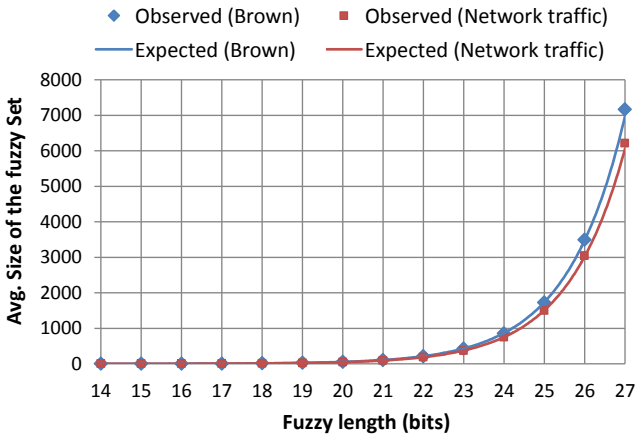


Fig. 4: The observed and expected sizes of fuzzy sets per fingerprint (32-bit) in Brown Corpus dataset (in blue) and network traffic (in red) with different fuzzy lengths.

We aim to show the trend of how the fuzzy-set sizes changes with the fuzzy length, which can be used to select the optimal fuzzy length used in the algorithm. We compute 32-bit fingerprints from the datasets, and then count the number of neighbors for each fingerprint. Fig. 4 shows the estimated and observed sizes of fuzzy sets for fuzzy lengths in the range of $[14, 27]$ for 218,652 and 189,878 fingerprints generated from the Brown Corpus dataset and the network traffic dataset. The figure shows that the empirical results observed are very close with the expected values of the fuzzy set sizes computed based on our analysis in Section IV. This close fit also indicates the uniform distribution of the fingerprints.

The fuzzy set is small when the fuzzy length is small, which is due to the sparsity nature of Rabin fingerprints. Given an estimated composition of traffic content, the data owner can use the result of this experiment to determine the optimal fuzzy length. In the datasets evaluated in the experiments, for fuzzy length of 26 and 27 bits, the K values are above 1,500 and 3,000, respectively. Because the data owner can defuzzify in POSTPROCESS very quickly, the false positives can be sifted out by the data owner. We also find that for a fixed fuzzy length the distribution of fuzzy-set sizes follows a Gaussian distribution. Different datasets may have different K size characteristics. We demonstrate the feasibility of estimating the fuzzy set sizes, which illustrates how fuzzy fingerprintings can be used to realize a privacy goal.

Summary Our detection rates in terms of the number of sensitive packets found do not decrease much with the decreasing size of disclosed fingerprint sets in Fig. 2, even when only 10% of the sensitive-data fingerprints are used for detection. Our experiments evaluate several noisy conditions such as *noise insertion* – `MediaWiki`-based leak scenario, and *data substitution* – for the keylogger- and `WordPress`-based leak scenarios. Our results indicate that our fingerprint filter can tolerate these three types of noises in the traffic to some degree. Our approach works well especially in the case where

consecutive data blocks are leaked (i.e., *local* data features are preserved). 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 it 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. We study the fuzzy set size and also verify the min-wise independence property of Rabin fingerprint, which are the building blocks of our fuzzy fingerprint method.

VI. Related Work

There have been several advances in understanding the privacy needs [25] or the privacy requirement of security applications [26]. In this paper, we identify the privacy needs in an outsourced data-leak detection service and provide a systematic solution to enable privacy-preserving DLD services.

Shingle with Rabin fingerprint [15] was used previously for identifying similar spam messages in a collaborative setting [27], as well as collaborative worm containment [28], virus scan [29], and fragment detection [30].

In comparison, we tackle the unique data-leak detection problem in an outsourced setting where the DLD provider is not fully trusted. Such privacy requirement does not exist in above models, e.g., the virus signatures are non-sensitive in the virus-scan paradigm [29]. We propose the fuzzy fingerprint approach to meet the special privacy requirement and present the first systematic solution to privacy-preserving data-leak detection with convincing results.

Most data-leak detection products offered by the industry, e.g., Symantec DLP [31], Global Velocity [32], Identity Finder [4], do not have the privacy-preserving feature and cannot be outsourced. GoCloudDLP [33] is a little different, which allows its customers to outsource the detection to a fully honest DLD provider. Our fuzzy fingerprint method differs from these solutions and enables its adopter to provide data-leak detection as a service. The customer or data owner does not need to fully trust the DLD provider using our approach.

Bloom filter [19] is a space-saving data structure for set membership test, and it is used in network security from network layer [34] to application layer [35]. The fuzzy Bloom filter invented in [36] constructs a special Bloom filter that probabilistically sets the corresponding filter bits to `1`'s. Although it is designed to support a resource-sufficient routing scheme, it is a potential privacy-preserving technique. We do not invent a variant of Bloom filter for our fuzzy fingerprint, and our fuzzification process is separate from membership test. The advantage of separating fingerprint fuzzification from membership test is that it is flexible to test whether the fingerprint is sensitive with or without fuzzification.

Besides fingerprint-based detection, other approaches can be applied to data-leak detection. If the sensitive data size is small and the patterns of all sensitive data are enumerable, string matching [37], [38] in network intrusion detection system (such as SNORT [39] or Bro [40]) can be used to detect data leaks. Privacy-preserving keyword search [41] or fuzzy keyword search [42] provide string matching approaches in semi-honest environments, but keywords usually do not cover enough sensitive data segments for data-leak detection.

Anomaly detection in network traffic can be used to detect data leaks. [5] detects any substantial increase in the amount of new information in the traffic, and entropy analysis is used in [43]. We present a signature-based model to detect data leaks and focus on the design that can be outsourced, thus the two approaches are different.

Another category of approaches for data-leak detection is tracing and enforcing the sensitive data flows. The approaches include data flow and taint analysis [6], legal flow marking [44], and file-descriptor sharing enforcement [8]. These approaches are different from ours because they do not aim to provide an remote service. However, pure network-based solution cannot handle maliciously encrypted traffic [45], and these methods are complementary to our approach in detecting different forms (e.g., encrypted) of data leaks.

Besides our fuzzy fingerprint solution for data-leak detection, there are other privacy-preserving techniques invented for specific processes, e.g., DNA matching [46], or for general purpose use, e.g., secure multi-party computation (SMC). Similar to string matching methods discussed above, [46] uses anonymous automata to perform comparison. SMC [47] is a cryptographic mechanism, which supports a wide range of fundamental arithmetic, set, and string operations as well as complex functions such as knapsack computation [48], automated trouble-shooting [49], network event statistics [50], [51], private information retrieval [52], genomic computation [53], private database query [54], private join operations [55], and distributed data mining [56]. The provable privacy guarantees offered by SMC comes at a cost in terms of computational complexity and realization difficulty. The advantage of our approach is its concision and efficiency.

VII. Conclusions and Future Work

We proposed fuzzy fingerprint, a privacy-preserving data-leak detection model and present its 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.

Acknowledgment

We would like to thank Michael Wolfe for his insightful comments and feedback.

References

[1] X. Shu and D. D. Yao, "Data leak detection as a service," in *Proceedings of the 8th International Conference on Security and Privacy in Communication Networks*, 2012, pp. 222–240.

[2] Risk Based Security, "Data breach quickview: An executives guide to 2013 data breach trends," February 2014, https://www.riskbasedsecurity.com/reports/2013-DataBreachQuickView.pdf, accessed October 2014.

[3] Ponemon Institute, "2013 cost of data breach study: Global analysis," May 2013, https://www4.symantec.com/mktginfo/whitepaper/053013_GL_NA_WP_Ponemon-2013-Cost-of-a-Data-Breach-Report_daiNA_cta72382.pdf, accessed October 2014.

[4] Identity Finder, "Discover sensitive data prevent breaches DLP data loss prevention," http://www.identityfinder.com/, accessed October 2014.

[5] K. Borders and A. Prakash, "Quantifying information leaks in outbound web traffic," in *Proceedings of the 30th IEEE Symposium on Security and Privacy*, 2009, pp. 129–140.

[6] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: capturing system-wide information flow for malware detection and analysis," in *Proceedings of the 14th ACM conference on Computer and Communications Security*, 2007, pp. 116–127.

[7] K. Borders, E. V. Weele, B. Lau, and A. Prakash, "Protecting confidential data on personal computers with storage capsules," in *Proceedings of the 18th USENIX Security Symposium*, 2009, pp. 367–382.

[8] A. Nadkarni and W. Enck, "Preventing accidental data disclosure in modern operating systems," in *Proceedings of the 20th ACM conference on Computer and Communications Security*, 2013, pp. 1029–1042.

[9] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna, "Revolver: An automated approach to the detection of evasive web-based malware," in *Proceedings of the 22nd USENIX Security Symposium*, 2013.

[10] X. Jiang, X. Wang, and D. Xu, "Stealthy malware detection and monitoring through VMM-based "out-of-the-box" semantic view reconstruction," *ACM Transactions on Information and System Security*, vol. 13, no. 2, p. 12, 2010.

[11] G. Karjoth and M. Schunter, "A privacy policy model for enterprises," in *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, June 2002, pp. 271–281.

[12] J. Jung, A. Sheth, B. Greenstein, D. Wetherall, G. Maganis, and T. Kohno, "Privacy oracle: a system for finding application leaks with black box differential testing," in *Proceedings of the 15th ACM conference on Computer and Communications Security*, 2008, pp. 279–288.

[13] Y. Jang, S. Chung, B. Payne, and W. Lee, "Gyrus: A framework for user-intent monitoring of text-based networked applications," in *Proceedings of the 23rd USENIX Security Symposium*, 2014, pp. 79–93.

[14] K. Xu, D. Yao, Q. Ma, and A. Crowell, "Detecting infection onset with behavior-based policies," in *Proceedings of the 5th International Conference on Network and System Security*, 2011, pp. 57–64.

[15] M. O. Rabin, "Fingerprinting by random polynomials," The Hebrew University of Jerusalem, Tech. Rep. TR-15-81, 1981.

[16] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *Journal of Computer and System Sciences*, vol. 60, no. 3, pp. 630–659, 2000.

[17] A. Z. Broder, "Some applications of Rabins fingerprinting method," in *Sequences II*. Springer, 1993, pp. 143–152.

[18] ——, "Identifying and filtering near-duplicate documents," in *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, 2000, pp. 1–10.

[19] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: A survey," *Internet mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[20] G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D. Thomas, and A. Zhu, "Anonymizing tables," in *Proceedings of the 10th International Conference on Database Theory*, 2005, pp. 246–258.

[21] R. Chen, B. C. Fung, N. Mohammed, B. C. Desai, and K. Wang, "Privacy-preserving trajectory data publishing by local suppression," *Information Sciences*, vol. 231, no. 0, pp. 83–97, 2013.

[22] M. O. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," Massachusetts Institute of Technology, Tech. Rep. MIT/LCS/TR-212, 1979.

[23] F. Liu, X. Shu, D. Yao, and A. R. Butt, "Privacy-preserving scanning of big content for sensitive data exposure with MapReduce," in *Proceedings of ACM CODASPY*, 2015.

[24] W. N. Francis and H. Kucera, "Brown corpus manual," *Brown University Department of Linguistics*, 1979.

[25] J. Kleinberg, C. H. Papadimitriou, and P. Raghavan, "On the value of private information," in *Proceedings of the 8th Conference on Theoretical Aspects of Rationality and Knowledge*, 2001, pp. 249–257.

[26] S. Xu, "Collaborative attack vs. collaborative defense," in *Collaborative Computing: Networking, Applications and Worksharing*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg, 2009, vol. 10, pp. 217–228.

[27] K. Li, Z. Zhong, and L. Ramaswamy, "Privacy-aware collaborative spam filtering," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 5, pp. 725–739, May 2009.

[28] M. Cai, K. Hwang, Y.-K. Kwok, S. Song, and Y. Chen, "Collaborative Internet worm containment." *IEEE Security & Privacy*, vol. 3, no. 3, pp. 25–33, May 2005.

[29] F. Hao, M. Kodialam, T. Lakshman, and H. Zhang, "Fast payload-based flow estimation for traffic monitoring and network security," in *Proceedings of the 2005 Symposium on Architecture for Networking and Communications Systems*, October 2005, pp. 211–220.

[30] L. Ramaswamy, A. Iyengar, L. Liu, and F. Douglis, "Automatic detection of fragments in dynamically generated web pages," in *Proceedings of the 13th International Conference on World Wide Web*, 2004, pp. 443–454.

[31] Symantec Data Loss Prevention, "Data loss prevention (DLP) software," http://www.symantec.com/data-loss-prevention/, accessed October 2014.

[32] Global Velocity Inc., "Cloud data security from the inside out," http://www.globalvelocity.com/, accessed October 2014.

[33] GTB Technologies Inc., "SaaS content control in the cloud," http://www.gtbtechnologies.com/en/solutions/dlp_as_a_service, accessed October 2014.

[34] S. Geravand and M. Ahmadi, "Survey Bloom filter applications in network security: A state-of-the-art survey," *Computer Networks*, vol. 57, no. 18, pp. 4047–4064, December 2013.

[35] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *Proceedings of the 33th IEEE Conference on Computer Communications*, 2014, pp. 2112–2120.

[36] C. P. Mayer, "Bloom filters and overlays for routing in pocket switched networks," in *Proceedings of the 5th International Student Workshop on Emerging Networking Experiments and Technologies*, 2009, pp. 43–44.

[37] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, 1975.

[38] P.-C. Lin, Y.-D. Lin, T. Lee, and Y. Lai, "Using string matching for deep packet inspection," *IEEE Computer*, vol. 41, no. 4, pp. 23–28, April 2008.

[39] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks," in *Proceedings of the 13th Conference on Systems Administration*, 1999, pp. 229–238.

[40] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[41] S. Ananthi, M. Sendil, and S. Karthik, "Privacy preserving keyword search over encrypted cloud data," in *Advances in Computing and Communications*, ser. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2011, vol. 190, pp. 480–487.

[42] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proceedings of 29th IEEE Conference on Computer Communications*, 2010, pp. 1–5.

[43] T. Fawcett, "ExFILD: A tool for the detection of data exfiltration using entropy and encryption characteristics of network traffic," 2010.

[44] J. Croft and M. Caesar, "Towards practical avoidance of information leakage in enterprise networks," in *Proceedings of the 6th USENIX Conference on Hot Topics in Security*, ser. HotSec'11, 2011, pp. 7–7.

[45] V. Varadharajan, "Internet filtering," *IEEE Security & Privacy*, vol. 8, no. 4, pp. 62–65, 2010.

[46] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik, "Privacy preserving error resilient DNA searching through oblivious automata," in *Proceedings of the 14th ACM conference on Computer and Communications Security*, 2007, pp. 519–528.

[47] A. C.-C. Yao, "How to generate and exchange secrets," in *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, 1986, pp. 162–167.

[48] D. Yao, K. B. Frikken, M. J. Atallah, and R. Tamassia, "Private information: To reveal or not to reveal," *ACM Transactions on Information and System Security*, vol. 12, no. 1, p. 6, 2008.

[49] Q. Huang, D. Jao, and H. J. Wang, "Applications of secure electronic voting to automated privacy-preserving troubleshooting," in *Proceedings of the 12th ACM conference on Computer and Communications Security*, 2005, pp. 68–80.

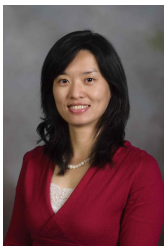
[50] P. Williams and R. Sion, "Usable PIR," in *Proceedings of the 13th Network and Distributed System Security Symposium*, 2008.

[51] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos, "Sepia: Privacy-preserving aggregation of multi-domain network events and statistics," in *Proceedings of the 19th USENIX Security Symposium*, 2010, pp. 15–15.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2015.2398363, IEEE Transactions on Information Forensics and Security

[52] X. Yi, R. Paulet, and E. Bertino, *Private Information Retrieval*, ser. Synthesis Lectures on Information Security, Privacy, and Trust. Morgan & Claypool Publishers, 2013.
[53] S. Jha, L. Kruger, and V. Shmatikov, "Towards practical privacy for genomic computation," in *Proceedings of the 29th IEEE Symposium on Security and Privacy*, 2008, pp. 216–230.
[54] X. Yi, M. G. Kaosar, R. Paulet, and E. Bertino, "Single-database private information retrieval from fully homomorphic encryption," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 5, pp. 1125–1134, May 2013.
[55] B. Carbunar and R. Sion, "Joining privately on outsourced data," in *Secure Data Management*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, vol. 6358, pp. 70–86.
[56] G. Jagannathan and R. N. Wright, "Privacy-preserving distributed k-means clustering over arbitrarily partitioned data," in *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, 2005, pp. 593–599.

Xiaokui Shu is a Ph.D. student in computer science at Virginia Tech and his research is in system and network security. He received his bachelor's degree from University of Science and Technology of China (USTC) in 2010 majoring in information security. Being the top student in the class, he graduated with Guo Moruo Award and was awarded SIMIT Chinese Academy of Sciences Scholarship in 2008. He assembled his first computer at 12. He succeeded at his first real-world penetration test at USTC and won the first prize in Virginia Tech Inaugural Cyber Security Summit Competition in 2011.



Danfeng (Daphne) Yao an associate professor in the Department of Computer Science at Virginia Tech, Blacksburg. She received her Computer Science Ph.D. degree from Brown University in 2007. Her recent research has been focused on network and system security and assurance, including anomaly detection in networked systems and programs, system integrity, malware program analysis, botnet detection, data exfiltration prevention. Professor Yao received the NSF CAREER Award in 2010 for her work on human-behavior driven malware detection, and most recently ARO Young Investigator Award for her semantic reasoning for mission-oriented security work in 2014. She received the Outstanding New Assistant Professor Award from Virginia Tech College of Engineering in 2012. Dr. Yao has several Best Paper Awards (ICICS '06, CollaborateCom '09, and ICNP '12). She was given the Award for Technological Innovation from Brown University in 2006. She held a U.S. patent for her anomaly detection technologies. Dr. Yao is an associate editor of *IEEE Transactions on Dependable and Secure Computing (TDSC)*. She serves as PC members in numerous computer security conferences, including ACM CCS. She has over 60 peer-reviewed publications.

Elisa Bertino is a professor at Purdue University Computer Science Department. She is the research director of CERIAS. Her research interests cover many areas in the fields of information security and database systems. Her work combines both theoretical and practical aspects, addressing applications on a number of domains, such as medicine and humanities. Professor Bertino is the editor-in-chief of *IEEE Transactions on Dependable and Secure Computing*. She also serves or has served on the editorial boards of several security journals, such as the *ACM Transactions on Information and System Security*, the *IEEE Security & Privacy Magazine*. Professor Bertino is an IEEE Fellow and a Fellow of ACM. She received the IEEE Computer Society Technical Achievement award in 2002 for outstanding contributions to database systems and database security and advanced data management systems, the 2005 Tsutomu Kanai Award by the IEEE Computer Society for pioneering and innovative research contributions to secure distributed systems, and most recently the 2014 ACM SIGSAC Outstanding Contributions Award for her seminal research contributions and outstanding leadership to data security and privacy for the past 25 years.