Data Leak Detection As a Service: Challenges and Solutions

Xiaokui Shu Danfeng (Daphne) Yao

Department of Computer Science, Virginia Tech

{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 require the data owner to 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* – 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. 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 the privacy, efficiency, accuracy and noise tolerance of our techniques. 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. It also indicates that the detection accuracy does not degrade when partial digests are used. We further provide a quantifiable method to measure the privacy guarantee offered by our fuzzy fingerprint framework.

Keywords privacy-preserving, data leak, detection, fingerprint, network security, algorithm

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 [5], *ii)* detecting stealthy malware with antivirus scanning or monitoring the host [21, 34, 39], and *iii)* enforcing policies to restrict the transfer of sensitive data. These approaches are complementary and can be deployed simultaneously. For example, the host-based solution described in storage capsule [5] prevents attackers from stealing data in the memory, which requires encryption and data-transfer rules, as well as complex operations to take host-wide snapshots. Most of the host-based solutions require the use of virtualization [21] or special hardware [34] to ensure the system integrity of the detector.

We present a novel *network-based* data-leak detection (DLD) solution that is both efficient and privacy-preserving. In comparison to host-based approaches, network-based data-leak detection focuses on analyzing the (unencrypted) content of outbound network packets for sensitive information. For example, a naive solution requires to inspect every packet for the occurrence of any of the sensitive data defined in the database. Such solutions generate alerts if the sensitive data is found in the outgoing traffic. However, this naive solution requires to store sensitive data in *plaintext* at the network interface, which is highly undesirable.

Another motivation for our privacy-preserving DLD work is cloud computing, which provides a natural platform for conducting data-leak detection by cloud providers as an add-on service. In cloud computing environments, an organization (data owner) may have already outsourced its services to a cloud provider, such as the email service for its own employees. The cloud provider may offer additional services such as inspecting email traffic for inadvertent data leak and serves as a DLD provider. This add-on DLD service requires minimal changes to the cloud provider's infrastructure and makes the cloud service more attractive.

However, *privacy* is a major roadblock for realizing outsourced data-leak detection. Conventional solutions require the data owner to reveal its sensitive data to the DLD provider. However, the DLD provider is always modeled as an honest-but-curious (aka semi-honest) adversary who is trusted to perform the inspection, but may attempt to learn about the data. 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-preserving data-leak detection system that enables the data owner to safely deploy locally, or to delegate the traffic-inspection task to DLD providers without exposing the sensitive data. 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 enable accurate comparison and detection. The DLD provider performs deep-packet inspection to identify whether these fingerprint patterns exist in the outbound traffic of the organization or not, according to a quantitative metric. We perform extensive experiments with real-world datasets in various data-leak scenarios to confirm the accuracy and efficiency of our proposed solutions.

Our technical contributions are summarized as follows.

1. We describe a novel fuzzy fingerprint method for detecting inadvertent data leak in network traffic. Its main feature is that the detection can be performed based on special digests without the sensitive data in plaintext, which minimizes the exposure of sensitive data during the detection. This strong privacy guarantee yields a powerful application of our fuzzy fingerprint method in the cloud computing environment, where the cloud provider can perform data-leak detection as an add-on service to its clients. We describe the quantitative privacy model, algorithms, and analysis in fuzzy fingerprint. The privacy model is useful beyond the specific fuzzy fingerprint problem studied. The detection is based on the fast set-intersection operation between the set of fingerprints generated from the payload of intercepted traffic (done by the DLD provider) and the set of fingerprints generated from the sensitive data (done by the data owner).

   We describe a realization of fuzzy fingerprint alternative to the above set-intersection approach, referred to by us as the *fingerprint filter*. Fingerprint filter uses a modified Bloom filter for membership testing, which we implement and evaluate for its performance in terms of running time. We further discuss the effect of collisions in Bloom filter on the privacy in our context of data-leak detection.

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 proposed 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 partial sensitive-data digests are used. In addition, these partial fingerprints fairly represent the full set of data without any bias.

2. Model and Requirements

There are several practical types of DLD deployment as shown below. Because of our digest-based detection method, sensitive data is not required to be disclosed in all three types of deployments. In what follows, we focus our description on the third distributed environments, which can be applied to the other two deployment scenarios.

1. To be deployed by an individual on a home network interface, or to be deployed by a company that provides network security services for home networks. The outsourcing home network security was also pointed out in [15].
2. To be deployed locally by an organization at the gateway router of its local area network.
3. To be deployed by a service provider that detects data leaks for an organization.

There are two technical challenges associated with network-based DLD detection. First, the DLD provider gains knowledge about the sensitive data when the traffic contains a leak. The challenge is how to restrict the degree of information that can be learned by the DLD provider in case of data leaks – the DLD provider has the access to the plaintext packet payload. The second challenge is how to make the detection noise-tolerant, for example, the intercepted packet payload may contain unrelated bytes or the sensitive data is truncated. We present our solutions to these challenges in Section 3 – we strategically randomize the detection for improving the privacy, and select local features to achieve the noise-tolerance. Next, we first give our privacy model specifying the adversary's capabilities and our assumptions. Then, we present our DLD framework including its key components and requirements.

2.1 Threat Model, Privacy Model, and Goal

Sensitive data may be leaked for several reasons. We aim to detect the inadvertent data leak in our threat model.

- Case I *Inadvertent data leakage*: 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 leaks may happen in different ways, e.g., 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 [22]).
- Case II *Malicious data leakage*: 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 are out of the scope of our network-based solution. Host-based defenses (such as detecting the infection onset [42]) 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.

There are two 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 provider may attempt to gain knowledge about the sensitive data. We model the DLD provider as a honest-but-curious adversary (aka semi-honest), who follows our protocols to carry out the operations, but may attempt to gain knowledge about the sensitive data.

The privacy goal in our fuzzy fingerprint mechanism is to prevent the DLD provider from inferring the exact knowledge of the sensitive data; the DLD provider is given the fingerprints of sensitive data and the content of network traffic which may or may not contain data leak. In our model, we aim to hide the sensitive values among other nonsensitive values, so that the DLD provider is unable to pinpoint sensitive data among them even under data-leak scenarios. We define our privacy goal as follows, following the K-anonymity privacy definition in the relational databases [32**?**]. K-anonymity has also been used in protecting routing privacy [38].

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 describe a novel fuzzy fingerprinting mechanism in the next section to improve the data protection against semi-honest DLD provider, by utilizing simple and effective randomization technique in fingerprint generation. 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.

2.2 Overview and Requirements for Privacy-Preserving DLD

A DLD provider performs the data-leak analysis on *transformed* and *randomized* inputs, and the transformation mechanism provides adequate privacy protection for the data owner. The 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.

The workflow in a network-based data-leak detection framework is as follows: DATA PRE-PROCESSING by the data owner, TRAFFIC PRE-PROCESSING AND DETECTION by the DLD provider, and ANALYSIS by the data owner. Data pre-processing is where the data owner takes the sensitive dataset and computes the corresponding set of digests. Traffic pre-processing and detection is where the DLD provider gathers network packets and inspects the content for data leaks. Analysis is where the data owner efficiently examines the alerts generated by the DLD provider, identifies and investigates the true leak instances and ignore false positives.

The key component in privacy-aware data-leak detection is the digest mechanism used. There are several requirements for such a digest algorithm: *one-wayness*, *noise tolerance*, and *subset independence* as explained below.

- *Onewayness:* Given a digest, it is computational hard to obtain the corresponding pre-image. The onewayness property partially provides the privacy protection for the data being analyzed (more discussion in Section 4).
- *Noise tolerance:* Similar inputs yield similar digests. For data-leak detection, the ability to tolerate certain degree of data transformation in traffic is particularly important. We use a sliding window to divide the original data into sets of shingles (i.e., pieces of data in q-grams), which can effectively *localize* the digest computation. Localization means the breakdown of the original data into pieces that are independent of each other. Fingerprints of shingles are then be computed.
- *Subset independence*: The partial digests (selected by the data owner to reveal to the DLD provider) are uniformly distributed across the entire dataset; any part of the original data is equally likely to appear in the partial digest set S_d. The requirement of subset independence is to ensure the fairness of sampling, and is especially useful if only partial digests are selected by the data owner and revealed to the DLD provider.

Next, we introduce shingle and Rabin fingerprint, based on which our data-leak detection framework is built, and explain how they satisfy the above requirements.

3. Details of Fuzzy Fingerprint Method

We describe the technical details of our fuzzy fingerprint mechanism for privacy-preserving data-leak detection, by first introducing shingle and Rabin fingerprint, and then presenting our randomization method for detection.

3.1 Shingle and Fingerprint

Noise tolerance is realized by us through the use of shingles. Shingles refer to the fixed size sequence of contiguous characters (i.e., q-gram). For example, for string `abcdefgh` the 3-gram shingle set consists of six elements {`abc`, `bcd`, `cde`, `def`, `efg`, `fgh`}. A sliding window is used in shingling a document, which can be viewed as taking *local* snapshots. The use of shingles for finding duplicate web documents was first appeared in [6, 8]. Shingling operation effectively breaks a dataset into multiple independent local pieces.

However, the use of shingles alone does not satisfy the onewayness requirement. One needs to transform each shingle element into its digest or fingerprint which uniquely represents the data. For this purpose, we use the Rabin fingerprint algorithm [29] which produces short and hard-to-reverse digests through the fast polynomial modulus operation. Rabin fingerprint has proven min-wise independence property.

Rabin fingerprints are computed as polynomial modulus operations, and can be implemented with fast XOR, shift, and table look-up operations. The shingle-and-fingerprint process is defined as follows. For a binary string $c_1 c_2 \ldots c_l$ of length l and an irreducible polynomial $p(x)$, we compute the fingerprint f_1 for the first k-bit shingle as follows.

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

We use a sliding window to generate shingles of k-bit long. For an input of size l, we repeat this computation to produce $l - k + 1$ shingles and their corresponding fingerprints.

In our scheme, the data owner reveals a subset of sensitive-data's fingerprints to the DLD provider for use in the detection by first sorting the fingerprints. The number of fingerprints to be selected is specified by the data owner. Rabin fingerprint is a special case of general linear permutation, which was proved to be *min-wise independent* [3, 9]. The property states that each input element is equally likely to become the minimum in the permuted set. This property ensures that the fingerprinting process gives an uniform and fair representation of the original data set, satisfying our subset independence requirement described in Section 2. We experimentally validate this property in the appendix, and we also analyze the privacy guarantees offered by our scheme in Section 4.

3.2 Detection With Fuzzy Fingerprints

A straightforward detection method is for the DLD provider to raise an alert if any of the sensitive fingerprints matches the fingerprints generated from the traffic. However, this exact-match approach has a privacy issue. In case of a data leak detected, there is a match between two fingerprints – one from the sensitive data and one from the 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 randomization 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 *fuzzy length* and *fuzzy set* as follows.

DEFINITION 3.1. *Given a fingerprint f, fuzzy length d is the number of the least significant bits in f that may be perturbed by the data owner, and d is less than the degree of the polynomial used to generate the fingerprint.*

DEFINITION 3.2. *Given a fuzzy length d, and a collection of fingerprints, the fuzzy set $S_{f,d}$ of a fingerprint f is the number of distinct fingerprints in the collection whose values differ from f by at most $2^d - 1$.*

The size of the fuzzy set $|S_{f,d}|$ is upper bounded by 2^d, but the actual size may be much smaller due to the sparsity of the fingerprint space and the (limited) size of the collection.

Next, we describe the operations in fuzzy fingerprinting using the above definitions. For simplicity, the description is based on one sensitive fingerprint, which can be easily generalized to multiple fingerprints.

1. FUZZIFY: This operation is run by the data owner. Given the fingerprint f of a shingle v and a fuzzy length d, the data owner flips an unbiased coin d times to generate the new least significant d bits in f. The rest of the bits in f are unchanged. The resulting fuzzy fingerprint f^* is released to the DLD provider for use in the detection.

2. DETECTION: This operation is run by the DLD provider. Given a fuzzy fingerprint f^* of some sensitive data and a fingerprint f' from the traffic, and a fuzzy length d, the DLD provider outputs 1 (indicating possible data leak) if values of f^* and f' differ by at most $2^d - 1$, and 0 otherwise. Because the fuzzy set of f^* includes the original fingerprint f, thus the true data leak can be detection (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. We provide deep analysis in later sections on fuzzy fingerprints including empirical results on the size of fuzzy set with real-world datasets.

   The range-based detection operation can be generalized to the membership testing with *Bloom filter* (through using fewer hash functions to increase collision probabilities), which we describe next and evaluate in Section 5.

   For all the data-leak matching instances (candidates) detected during the range-based detection, the DLD provider outputs the set of $\{(x_1, f_1), \ldots, (x_i, f_i), \ldots)\}$ pairs to the data owner, where x is the shingle appearing in the traffic, and f is its Rabin fingerprint.

3. DEFUZZIFY: This operation is run by the data owner. Given the data-leak instance candidates represented by a set of tuples $\{(x_1, f_1), (x_2, f_2), \ldots\}$, the data owner searches to see if the sensitive data's fingerprint f exists. If there exists a $f_i = f$ and $x_i = v$, then there is a true data leak, otherwise the submitted candidates can be safely ignored by the data owner.

The advantage of the above 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 it can quickly distinguish true and false leak instances. Given the digest f of a piece of sensitive data, a large collection D of traffic fingerprints, and a positive integer $K \ll |D|$, the data owner can choose a fuzzy length 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 d value satisfying the privacy requirement) is important for the efficiency of the DEFUZZIFY operation. Due to the dynamic nature of network traffic, d needs to be estimated accordingly. We provide quantitative analysis in later sections on fuzzy fingerprints including empirical results on the sizes of fuzzy sets.

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). However, one issue of this naive 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.

3.3 Extensions: Fingerprint Filter and Bit Mask

Fingerprint Filter Bloom filter [7, 14] is a well-known data structure for performing set-membership test, and has the advantage of space saving. 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. A mismatched indicates that v does not belong to the set, otherwise, v may be a set member with a probability based on the number of hash functions and their properties.

Bloom filter in combination with Rabin fingerprints is referred to by us as the *fingerprint filter*. For our fingerprint filter, we replace the original hash function (e.g., SHA-1) with Rabin fingerprint. Our detection techniques described in this work is order oblivious, i.e., the order of fingerprints is not considered. Our network-based technique is complementary to any host-based data-leak detection solutions.

Bit mask We can also generalize the FUZZIFY operation with a bit mask, which specifies any arbitrarily chosen d bits for comparison. The d bits used by the data owner in FUZZIFY can be any arbitrary bits in the original fingerprint (as opposed to the least significant bits), as long as:

- The DLD provider and the data owner agree on the bit mask used.
- The Hamming distance between a binary fingerprint and its fuzzy version is bounded by d.

The DETECTION operation needs to be generalized accordingly. Instead of identifying matching fingerprints in the specific range of a fuzzy fingerprint, the DLD provider creates a bit vector of unperturbed bits and sets the perturbed bit positions to be *Don't Care* bits. Any fingerprint that matches this special bit vector is recorded by the DLD provider and reported to the data owner. Specifically, we formalize this generalized detection mechanism based on the following Hamming distance measure $H(f, g)$ between fingerprints f and g in their binary representations. The detection records fingerprints from the traffic whose Hamming distances to this special bit vector are 0.

$$H(f,g) = \begin{cases} 1 & f_i \neq g_i \\ 0 & f_i = g_i \\ 0 & f_i \text{ or } g_i \text{ is a don't-care bit} \end{cases}$$

4. Analysis and Discussion

Storing digests as opposed to the original data on DLD providers defends the data confidentiality against not only curious providers, but also security breaches to their servers caused by outside attackers. We analyze the security and privacy guarantees provided by our data-leak protection 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 the sensitive data. The fingerprint-based privacy relies on the *one-wayness* of the fingerprint computation. It is difficult to guess the original shingles with only the knowledge of fingerprints.

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^d, where d is the fuzzy length. For a large shingle set of size $2^{m-d} \leq n \leq 2^m$, the expected value of $K = \frac{n}{2^m} \times 2^d$, assuming that the fingerprints of shingles are uniformly distributed. 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 4.1. We summarize the above privacy analysis in the following theorem.

THEOREM 4.1. *A polynomial-time adversary has no greater than $\frac{2^{m-d}}{n}$ probability of correctly inferring a sensitive shingle, where m is the length of the fingerprint in bits, d is the fuzzy length, and $n \in [2^{m-d}, 2^m]$ 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 given the following values: the total number of sensitive fingerprints M, the expected size K of a fuzzy set of fuzzy length d, the percentage α of sensitive fingerprints revealed to the DLD provider (the set of partial sensitive fingerprints selected denoted by S_d), and the expected rate β of the leak in terms of the percentage of sensitive fingerprints in S_d that may appear in the network traffic. Based on Theorem 4.1, $K = \frac{n}{2^{m-d}}$. Let n be the size of traffic fingerprints, 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.

$$R = \frac{\alpha\beta MK}{n} = \frac{\alpha\beta M}{2^{m-d}} \tag{1}$$

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 [8], 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 [6] and are omitted.

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.

Our proposed method is suitable for detecting sensitive data units such as code words, social security numbers, credit card numbers, or a large sensitive database where any consecutive segments leaked. Selecting a subset of the sensitive fingerprints (i.e., partial fingerprints) to disclose may result in false negatives – the leaked data may evade the detection because it is not covered by the partial fingerprints. This issue illustrates the tradeoff between detection accuracy and privacy guarantee. Our experiments evaluate continuous data segments being leaked.

Prior knowledge of sensitive data Theorem 4.1 assumes the uniform likelihood of being sensitive for all fingerprints in a fuzzy set. However, the adversary may be able to differentiate corresponding shingles based on the auxiliary information regarding the sensitive data (e.g., the format or space of the data). For example, a strong password may look different from a regular English word. Although this observation is true, such a differentiation analysis is difficult because of the diversity of traffic, since the serialized binary data (e.g., image or video) may appear as random as strong passwords.

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.

5. Implementation and 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 same padding scheme mentioned in [28] 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 with several virtual machines 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 gives it to a DLD server (Linux). Using the pre-extracted 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.

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

$$S_{packet} = \frac{|\mathbb{F}^D_{sens} \cap \mathbb{F}_{packet}|}{min(|\mathbb{F}^A_{sens}|, |\mathbb{F}_{packet}|)} \times \frac{|\mathbb{F}^A_{sens}|}{|\mathbb{F}^D_{sens}|}$$

$\mathbb{F}_{packet}$ is the set of all fingerprints extracted in a packet. $\mathbb{F}^A_{sens}$ is the set of all sensitive fingerprints, and $\mathbb{F}^D_{sens}$ is the set of sensitive fingerprints used in the detection. The users in the local network (data owner) compute $\mathbb{F}^{A}_{sens}$ and reveal $\mathbb{F}^{D}_{sens}$ ($\mathbb{F}^{D}_{sens} \subseteq \mathbb{F}^{A}_{sens}$) to the DLD server (DLD provider). 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 and trigger alerts.

In this section, 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's fingerprint?

The questions are experimentally addressed in our following sections with the last one answered in the appendix.

5.1 Shingling and Fingerprinting Accuracy Evaluation

We design a group of experiments to evaluate how well our underlying shingling and fingerprinting strategy works in data-leak detection. We generate 20,000 personal financial records as the sensitive data, which contain *person name*, *social security number*, *credit card number*, *credit card expiration date*, and *credit card CVV* (samples in Table 1), and store them in a text file.

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 The outbound HTTP traffic of Internet-surfing by 20 users are captured (30 minutes per user), and given to the DLD server to analyze. No sensitive data should be detected, since it is not provided to the users.

Exp.3 The Enron dataset (collected and prepared by the CALO Project: 2.6 GB data, 150 users' 517424 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 should be detected, neither.

All sensitive fingerprints ($\mathbb{F}^{D}_{sens} = \mathbb{F}^{A}_{sens}$) are used in the detection, and the results are shown in Table 2.

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

Name	SSN	Credit Card	Exp Date	CVV
Kally	197908573	5247328478604466	02/2012	066
Vince	174257302	5948967605207190	12/2013	048
Faydra	301788837	2208159421142290	02/2013	792
Alisha	499197573	8481168228544639	03/2012	000
Yoshiko	993195251	9425620913759297	11/2012	655

Table 1. Samples of the sensitive data enties: social security numbers and credit card information

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 2. 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.

value is less than one, because the layered headers (IP, TCP, HTTP, etc.) in a packet are not sensitive.

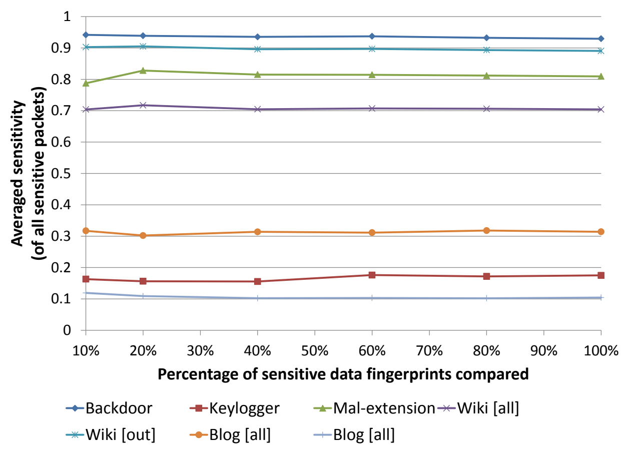
The next two experiments are designed to estimate the false positive rate. In these two experiments, none of the packets has a sensitivity value greater than 0.05, and the average sensitivity is very low as shown in Table 2.

This group of experiments draws a strong conclusion that our algorithm performs well with high true positives and low false alarms in data-leak detection on plaintext. A threshold of sensitivity can be easily set to distinguish sensitive packets from normal ones. Analysis of our system using partial fingerprints is discussed in Section 5.2.

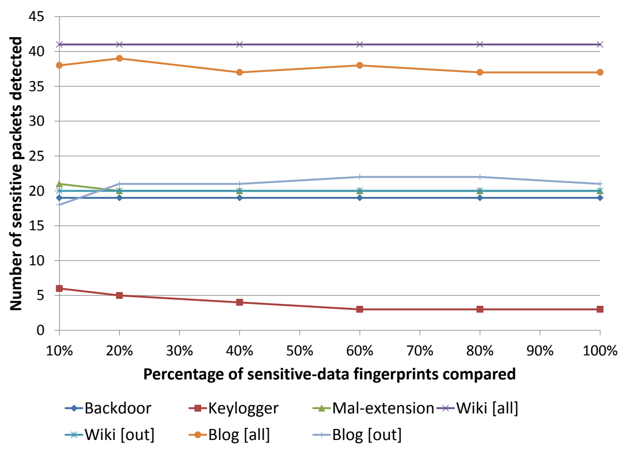
5.2 Data-leak Detection With Partial Fingerprints

The advantages of using partial fingerprints are good privacy control and efficient processing. We evaluate this situation where only a (small) portion of sensitive data's fingerprints are revealed to the DLD server for detection. We are particularly interested in measuring the percentage of partial 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 emulate 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 browser extension *FFsniFF* records the information in sensitive web forms, and emails the data to a SMTP server.
- In the keylogging scenario, a keylogger *EZRecKb* exports intercepted keystroke values on a user's machine (Win-



(a)



(b)

Figure 1. 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.

dows 7). 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, and the sensitivity threshold is set $\mathcal{S}_{thres} = 0.05$. All packets with $\mathcal{S}_{packet} \geq \mathcal{S}_{thres}$ are marked as sensitive, which trigger alerts and are counted in computing average sensitivity.

Figure 1 shows the comparison of performance across various size of partial 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 the defuzzification. The results show that the use of partial sensitive-data fingerprints does not degrade the detection rate compared to the use of full sets of sensitive-data fingerprints. Our rationale for this observation is as follows: For 8-byte shingle, each shingle gathers 8 bytes information around it. In other words, 10% of fingerprints cover up to 80% of sensitive data.

In Figure 1 (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 1 (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.

5.3 Runtime Comparison between Bloom Filter and Fingerprint Filter

Our fingerprint filter implementation is based on the Bloom filter library in Python (`Pybloom`). We make a comparison between 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 2 – 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 2. 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 number of hash functions used in Bloom filters does not significantly impact their runtime. The Bloom filter with 2 hash functions is the slowest in Figure 2. We speculate that the slowdown is due to the high number of collisions, whereas with more hash functions (and fewer collisions) fingerprints can be ruled out quickly *without* completing the full bit-vector comparison.

Using fewer hash functions in Bloom filters or fewer polynomials in the fingerprint filters produces more false positives at the DLD provider. The (true and false) positive instances are reported to the data owner who can then quickly identifies the real data leaks, similar to the DE-

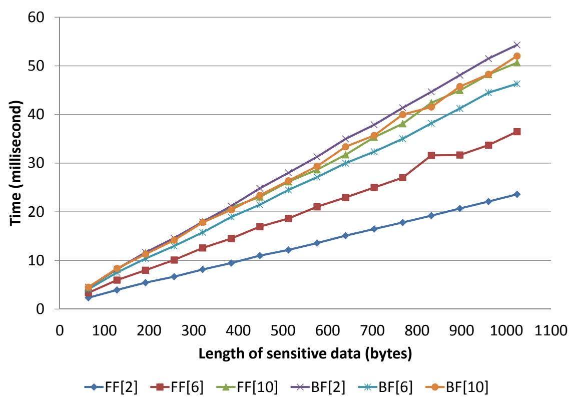


Figure 2. The overhead of using the filters to detect data-leak, the runtime is per packet (averaged from 100 packets) against all 10 pieces of sensitive data, and the X-axis indicates the amount of sensitive information in a packet.

FUZZIFY operations described in Section 3.2. 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%. Our fuzzy fingerprint should not be confused with fuzzy Bloom filter [27].

5.4 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 evaluate empirically the average size of the fuzzy set associated with a given fuzzy length with both Brown Corpus (English text) and network traffic (composed of Internet-surfing traffic of a user). We aim to show the trend of how 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, sort the fingerprints, and compute the number of neighbors for each fingerprint. Figure 3 shows the estimated and observed sizes of fuzzy sets for fuzzy lengths in the range of $[14, 27]$ for 218,652 fingerprints generated from the Brown Corpus dataset, and 189,878 fingerprints from a network traffic dataset. The Y-axis reflects on how many neighbors each fingerprint has, given a range defined by the fuzzy length. Figure 3 shows that the empirical results observed are very close with the expected values of the fuzzy set sizes computed based on Theorem 4.1. This close similarity 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. The data owner may use this type of experiments to determine the optimal fuzzy length, given an estimated composition of traffic content. 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 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 (not shown). K values may be set differently for different sensitive fingerprints. Other datasets may have different size characteristics. What is important in our experiment is the demonstration of the feasibility of estimating the fuzzy set sizes, which illustrates how fuzzy fingerprinting can be used to realize our privacy goal.

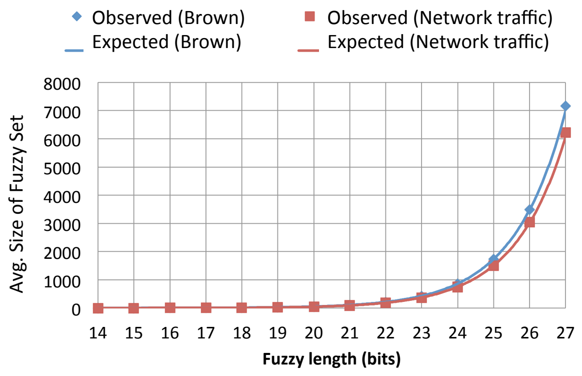


Figure 3. The observed and expected sizes of fuzzy sets per fingerprint (32-bit) in Brown Corpus dataset (in blue) and network traffic (in red) with different fuzzy lengths.

Summary We develop an accurate privacy preserving data-leak detection strategy. Table 2 shows the low false positive and high true positive properties of our strategy.

Our detection rates in terms of the number of sensitive packets found do not decrease much with the decreasing size of partial fingerprint sets in Figure 1, 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.

From the implementation profiling, we also find out that the speed of our system is heavily restrained by Python virtual machine, especially due to the overhead of Python function calls, though we use C/C++ extension to compute Rabin's fingerprint and SHA-1. We plan to rewrite our whole framework in C/C++ in the future for faster performance.

Encrypted traffic, which cannot be directly inspected [37], 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 [46] can be integrated.

6. Related Work

There have been several advances in developing *privacy-aware* collaborative solutions from both system [11, 25, 33] and theory perspectives [23, 43]. Specifically, Rabin fingerprint [29] based on shingles was used previously for identifying similar spam messages in a collaborative setting [25], as well as collaborative worm containment [11], virus scan [17], Web template detection [2], and fragment detection [30].

Our work fundamentally differs from the above shingle-based studies [11, 17] 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 [17], 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 [4]. 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.

The method of deep packet inspection is also widely used in network intrusion detection system (NIDS), such as SNORT [31] and Bro. They focus on designing and implementing efficient string matching algorithms [1, 24] to handle short and flexible patterns in network traffic [26]. 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 automata) 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.

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 [44], 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 such as private set intersection [16, 47], as well as complex functions such as knapsack computation [45], automated trouble-shooting [18], peer computation [13], network event statistics [10], private information retrieval [40, 41], genomic computation [20], private join operations [12], and distributed data mining [19]. 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

Preventing sensitive data from being compromised is an important and practical research problem. We proposed a novel fuzzy fingerprint framework and algorithms to realize privacy-preserving data-leak detection. Using special digests, the exposure of the sensitive data is kept to a minimum during the detection. We described its application in the cloud computing environments, where the cloud provider naturally serves as the DLD provider. We defined our privacy goal by quantifying and restricting the probability that the DLD provider identifies the exact value of the sensitive data. We presented the protocols and data structures including a Bloom-filter based fuzzy fingerprint filter. Our extensive experiments validate the accuracy, privacy, and efficiency of our solutions. For future work, we will test our current solution on binary sensitive data, and then focus on designing solutions that will efficiently prevent the leakage of complex data types, especially dynamically-changing sensitive data, such as source code of programs and sensitive documents constantly being modified.

References

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

[2] Bar-Yossef, Z., and Rajagopalan, S. Template detection via data mining and its applications. In *Proceedings*

of the 11th International World Wide Web Conference (WWW) (May 2002).

[3] Bohman, T., Cooper, C., and Frieze, A. M. Min-wise independent linear permutations. *Electr. J. Comb. 7* (2000).

[4] Borders, K., and Prakash, A. Quantifying information leaks in outbound web traffic. In *Proceedings of the IEEE Symposium on Security and Privacy* (May 2009).

[5] 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.

[6] Broder, A. Some applications of Rabins fingerprinting method. *Sequences II: Methods in Communications, Security, and Computer Science* (1993), 143–152.

[7] Broder, A., and Mitzenmacher, M. Network applications of Bloom filters: A survey. In *Internet Mathematics* (2002).

[8] 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.

[9] 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.

[10] 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).

[11] 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.

[12] 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.

[13] Duan, Y., Youdao, N., Canny, J., and Zhan, J. P4P: Practical large-scale privacy-preserving distributed computation robust against malicious users. In *Proceedings of USENIX Security* (2010).

[14] Erdogan, O., and Cao, P. Hash-AV: fast virus signature scanning by cache-resident filters. *IJSN 2*, 1/2 (2007), 50–59.

[15] Feamster, N. Outsourcing home network security. In *HomeNets* (2010), pp. 37–42.

[16] Freedman, M., Nissim, K., and Pinkas, B. Efficient private matching and set intersection. In *Advances in Cryptology – Eurocrypt '04* (May 2004), vol. 3027 of *LNCS*, Springer-Verlag, pp. 1–19.

[17] 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.

[18] 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).

[19] 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), pp. 593–599.

[20] Jha, S., Kruger, L., and Shmatikov, V. Towards practical privacy for genomic computation. In *IEEE Symposium on Security and Privacy* (2008), IEEE Computer Society.

[21] 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).

[22] 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).

[23] 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.

[24] Kumar, S., Dharmapurikar, S., Yu, F., Crowley, P., and Tumer, J. S. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In *Proceedings of the ACM SIGCOMM 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (2006).

[25] Li, K., Zhong, Z., and Ramaswamy, L. Privacy-aware collaborative spam filtering. *IEEE Transactions on Parallel and Distributed systems 20*, 5 (May 2009).

[26] Lin, P., Lin, Y., Lai, Y., Lee, T. and Lee, T. Using string matching for deep packet inspection. *IEEE Computer* (2008).

[27] 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).

[28] Rabin, M. O. Digitalized signatures as intractable as factorization. Tech. Rep. MIT/LCS/TR-212, MIT Laboratory for Computer Science, Jan. 1979.

[29] Rabin, M. O. Fingerprinting by random polynomials. Tech. rep., Center for Research in Computing Technology, Harvard University, 1981. TR-15-81.

[30] 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).

[31] Roesch, M. Snort: lightweight intrusion detection for networks In *Proceedings of the 13th Conference on Systems Administration (LISA-99)* (1999).

[32] Samarati, P. Protecting respondents' identities in microdata release. *IEEE Trans. Knowl. Data Eng.* (2001).

[33] SARWAR, B., KARYPIS, G., KONSTAN, J., AND RIEDL, J. Item-based collaborative filtering recommendation algorithms. In *International World Wide Web Conference Proceedings of the 10th international conference on World Wide Web* (2001).

[34] 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).

[35] SWEENEY, L. K-anonymity: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* (2002).

[36] TRONCOSO-PASTORIZA, J. R., KATZENBEISSER, S., AND CELIK, M. U. Privacy preserving error resilient DNS Searching through oblivious automata. In *ACM Conference on Computer and Communications Security* (2007), P. Ning, S. D. C. di Vimercati, and P. F. Syverson, Eds., ACM, pp. 519–528.

[37] VARADHARAJAN, V. Internet filtering issues and challenges. *Journal of IEEE Security & Privacy* (2010), 62 – 65.

[38] WANG, P., NING, P., AND REEVES, D. S. A k-anonymous communication protocol for overlay networks. In *ASIACCS* (2007), F. Bao and S. Miller, Eds., ACM, pp. 45–56.

[39] 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).

[40] WILLIAMS, P., AND SION, R. Usable PIR. In *NDSS* (2008), The Internet Society.

[41] 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).

[42] 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).

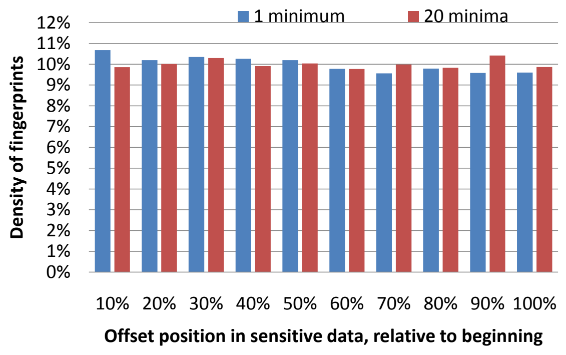
[43] XU, S. Collaborative attack vs. collaborative defense. In *The 4th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)* (2008).

[44] 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.

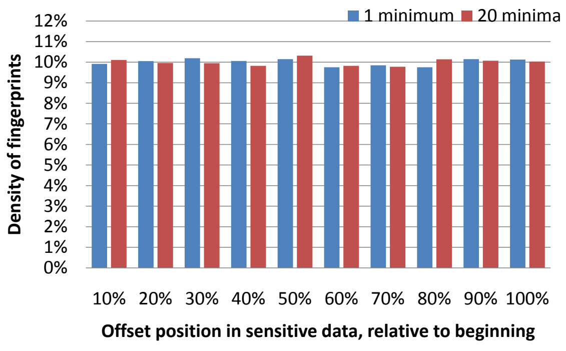
[45] YAO, D., FRIKKEN, K. B., ATALLAH, M. J., AND TAMASSIA, R. Private information: to reveal or not to reveal. *ACM Trans. Inf. Syst. Secur. 12*, 1 (2008).

[46] 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).

[47] ZHOU, X., DING, X., AND CHEN, K. Lightweight delegated subset test with privacy protection. *In Proceedings of the Information Security Practice and Experience (ISPEC)* (2011).



(a)



(b)

Figure 4. The distribution of the smallest subset of fingerprints in terms of their positions in the original data in random text (a) and Brown Corpus (b).

A. Appendix: Verification on Min-Wise Independence Property

The subset independence property provides the strong assurance on the fairness of using partial fingerprints in our detection. This property is important because the DLD provider is only allowed to access a subset of sensitive-data fingerprints for detection. To ensure the smallest subset of fingerprints (used for detection) are uniformly distributed across the entire set of sensitive fingerprints, we perform several experiments as follows.

We plot the distribution of the smallest subset of fingerprints in terms of their positions in the original data. The position is calculated as the percentage offset relative to the beginning, e.g., the first fingerprint is 0% and the last one 100%. In the first experiment, we select the smallest fingerprint and the second experiment, we select the 20 smallest fingerprints. The results shown in Figure 4 (a) are averaged over 10,000 and 500 runs on 1KB randomly generated data, respectively. The data used to compute shingles and fingerprints is 1KB. Besides random bytes, we also test the Brown Corpus in Figure 4 (b). Both results validate the min-wise independence property of Rabin fingerprint, as the smallest fingerprints are uniformly distributed across the data.