ISSN 2395-695X (Print)
ISSN 2395-695X (Online)

International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)
Vol. 2, Issue 4, April 2016

AN AUTOMATED APPROACH FOR SECURING SENSITIVE DATA WITH ADVANCED DLD AND DLP

J.JESSIMA[1], V.BALAMURUGAN[2]

(Anna Univ Affiliated) Master of Engineering, Department of Computer Science and Engineering[1],
(Anna Univ Affiliated) Associate Professor,Department of Computer Science and Engineering[2] *Mohamed Sathak Engineering College, Kilakarai, Ramanathapuram dist., TamilNadu*

[1]spjjessima@gmail.com

[2]vbalram78@gmail.com

Abstract— **Detection and prevention of data leakage is the major issue since attacks plays an important key role during data transmission. In existing paper they proposed and present a privacy preserving data-leak detection (DLD) solution to solve the issue where a special set of sensitive data digests is used in detection. An existing system used only fuzzy fingerprint technique that enhances data privacy during data-leak detection operations. When using fuzzy fingerprint increases the computational complexity as well as increases the memory to store the keys. So our proposed system is going to introduce a new encryption technique which is SIMON-SPECK encryption algorithm it will carry a small size of keys and their parameters with very small in range. It reduces the computational complexity and reduces the key size small. Since, SIMO-SPECK only uses small key size at the same time provide security provided of RSA encryption algorithm. Finally our simulation result shows that our proposed system reduces the time as well as improves system accuracy.**

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

I. Introduction

Network security consists of the policies adopted to prevent and monitor authorized access, misuse, modification, or denial of a computer network and network-accessible resources. Network security involves the authorization of access to data in a network, which is controlled by the network administrator. Users choose or are assigned an ID and password or other authenticating information that allows them access to information and programs within their authority. Network security covers a variety of computer networks, both public and private, that are used in everyday jobs; conducting transactions and communications among businesses, government agencies and individuals. Networks can be private, such as within a company, and others which might be open to public access. Network security is involved in organizations, enterprises, and other types of institutions. It does as its title explains: It secures the network, as well as protecting and overseeing operations being done.

The most common and simple way of protecting a network resource is by assigning it a unique name and a corresponding password. Network security starts with authenticating, commonly with a username and a password. Since this requires just one detail authenticating the user name —i.e., the password— this is sometimes termed one-factor authentication. With two-factor authentication, something the user 'has' is also used (e.g., a security token or 'dongle', an ATM card, or a mobile phone); and with three-factor authentication, something the user 'is' also used (e.g., a fingerprint or retinal scan). Once authenticated, a firewall enforces access policies such as what services are allowed to be accessed by the network users. Though effective to prevent unauthorized access, this component may fail to check potentially harmful content such as computer worms or Trojans being transmitted over the network.

Anti-virus software or an intrusion prevention system (IPS)[2] help detect and inhibit the action of such malware. An anomaly-based intrusion detection system may also monitor the network like wireshark traffic and may be logged for audit purposes and for later high-level analysis. Communication between two hosts using a network may be encrypted to maintain privacy. Honeypots, essentially decoy network-accessible resources, may be deployed in a network as surveillance and early-warning tools, as the honeypots are not normally accessed for legitimate purposes. Techniques used by the attackers that attempt to compromise these decoy resources are studied during and after an attack to keep an eye on new exploitation techniques. Such analysis may be used to further tighten security of the actual network being protected by the honeypot. A honeypot can also direct an attacker's attention away from legitimate servers. A honeypot encourages attackers to spend their time and energy on the decoy server while distracting their attention from the data on the real server. Similar to a honeypot, a honeynet is a network set up with intentional vulnerabilities.

Its purpose is also to invite attacks so that the attacker's methods can be studied and that information can be used to increase network security. A honeynet typically contains one or more honeypots. Security management for networks is different for all kinds of situations. A home or small office may only require basic security while large businesses may require high-maintenance and advanced software and hardware to prevent malicious attacks from hacking and spamming. Networks are subject to attacks from malicious sources. Attacks can be from two categories: "Passive" when a network intruder intercepts data traveling through the network, and "Active" in which an intruder initiates commands to disrupt the network's normal operation.

Intrusion detection System

An intrusion detection system (IDS) is a device or software application that monitors network or system activities for malicious activities or policy violations and produces reports to a management station. IDS come in a variety of “flavors” and approach the goal of detecting suspicious traffic in different ways. There are network based (NIDS) and host based (HIDS) intrusion detection systems. NIDS is a network security system focusing on the attacks that come from the inside of the network (authorized users). When we classify the designing of the NIDS according to the system interactivity property, there are two types: on-line and off-line NIDS. On-line NIDS deals with the network in real time and it analyses the Ethernet packet and applies it on the some rules to decide if it is an attack or not. Off-line NIDS deals with a stored data and pass it on a some process to decide if it is an attack or not.

Some systems may attempt to stop an intrusion attempt but this is neither required nor expected of a monitoring system. Intrusion detection and prevention systems (IDPS) are primarily focused on identifying possible incidents, logging information about them, and reporting attempts. In addition, organizations use IDPSes for other purposes, such as identifying problems with security policies, documenting existing threats and deterring individuals from violating security policies. IDPSes have become a necessary addition to the security infrastructure of nearly every organization. IDPSes typically record information related to observed events notify security administrators of important observed events and produce reports. Many IDPSes can also respond to a detected threat by attempting to prevent it from succeeding.

They use several response techniques, which involve the IDPS stopping the attack itself, changing the security environment (e.g. reconfiguring a firewall) or changing the attack's content. A "network intrusion detection system (NIDS)" monitors traffic on a network looking for suspicious activity, which could be an attack or unauthorized activity. A large NIDS server can be set up on a backbone network, to monitor all traffic; or smaller systems can be set up to monitor traffic for a particular server, switch, gateway, or router. In addition to monitoring incoming and outgoing network traffic, a NIDS server can also scan system files looking for unauthorized activity and to maintain data and file integrity. The NIDS server can also detect changes in the server core components.

In addition to traffic monitoring, a NIDS server can also scan server log files and look for suspicious traffic or usage patterns that match a typical network compromise or a remote hacking attempt. The NIDS server can also server a proactive role instead of a protective or reactive function. Possible uses include scanning local firewalls or network servers for potential exploits, or for scanning live traffic to see what is actually going on. Keep in mind that a NIDS server does not replace primary security such as firewalls, encryption, and other authentication methods. The NIDS server is a backup network integrity device. Neither system (primary or security and NIDS server) should replace common precaution (building physical security, corporate security policy, etc.).

An intrusion detection system (IDS) monitors network traffic and monitors for suspicious activity and alerts the system or network administrator. In some cases the IDS may also respond to anomalous or malicious traffic by taking action such as blocking the user or source IP address from accessing the network. IDS come in a variety of “flavors” and approach the goal of detecting suspicious traffic in different ways.

IDS was originally developed this way because at the time the depth of analysis required for intrusion detection could not be performed at a speed that could keep pace with components on the direct communications path of the network infrastructure. As explained, the IDS are also a listen-only device. The IDS monitors traffic and report its results to an administrator, but cannot automatically take action to prevent a detected exploit from taking over the system. Attackers are capable of exploiting vulnerabilities very quickly once they enter the network, rendering the IDS an inadequate deployment for prevention device.

NIDS

Network Intrusion Detection Systems are placed at a strategic point or points within the network to monitor traffic to and from all devices on the network. Ideally you would scan all inbound and outbound traffic; however doing so might create a bottleneck that would impair the overall speed of the network. An NIDS is strategically positioned at various points on the network to monitor traffic going to and from network devices. NIDS solutions offer sophisticated, real-time intrusion detection capabilities often consisting of an assembly of interoperating pieces: a standalone appliance, hardware sensors, and software components are typical components that make up an NIDS. These pieces working in concert allow for a wider range of network intrusion detection capabilities than HIDS solutions.

Data Leakage Detection

Sometimes a data distributor gives sensitive data to a set of third parties. Sometime later, some of the data is found in an unauthorized place (e.g., on the web or on a user's laptop). The distributor must then investigate if data leaked from one or more of the third parties, or if it was independently gathered by other means. 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.

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.

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.

1) *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.

2) *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.

A 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 for-warding 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

1

, 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.

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, Our solution confines the amount of maximal information learned during the detection and provides quantitative guarantee for data privacy.

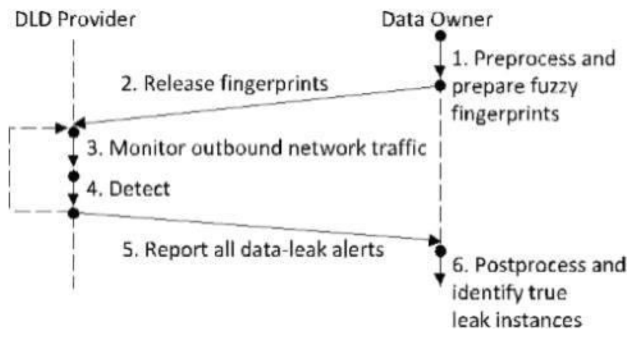

DLD Provider
Data Owner
1. Preprocess and prepare fuzzy fingerprints
2. Release fingerprints
3. Monitor outbound network traffic
4. Detect
5. Report all data-leak alerts
6. Postprocess and identify true leak instances

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

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 inde-pendence property [16], which supports fast random finger-prints 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] and [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 $p_f + 1$.

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

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. 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.

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 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 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.

C. Extensions

1) 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.

2) 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 S* to perform partial disclosure. The r elements are then sent to the DLD provider in RELEASE operation instead of 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.

A. 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 finger-print 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 finger-prints, 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 finger-prints. 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 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 provide a sensitive shingle (case i.a).

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

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 finger-prints 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.

$$S_{packet} = \frac{|\ddot{S}^* \gg p_d \cap T \gg p_d|}{min(|\ddot{S}^*|, |T|)} \times \frac{|S^*|}{|\ddot{S}^*|} \qquad (5)$$

T is the set of all fingerprints extracted in a packet. S^* is the set of all sensitive fuzzy fingerprints. For each piece of sensitive data, the data owner computes S^* and reveals a partial set $\ddot{S}^*$ ($\ddot{S}^* \subseteq S^*$) 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) $|\ddot{S}^*| / |S^*|$ estimates the leaking level. 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 S_{packet} ($S_{packet} \in [0, 1]$). If it is higher than a threshold $S_{thres} \in (0, 1)$, T is reported back to the data owner, and the data owner uses (6) to determine whether it is a real leak in Postprocess.

$$S_{packet} = \frac{|S \cap T|}{min(|S|, |T|)} \qquad (6)$$

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

The use of S_{packet} and 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. S_{packet} is an accumulated score and 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 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.

1) Simple Leaking Scenarios: We test our prototype without partial disclosure in simple leaking scenarios, i.e., $\ddot{S}^* = S^*$. 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*.

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.

2) 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[3] intercepted keystroke values on a user's host. It con-nects 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 disclo-sure rate. The subset of the sensitive fingerprints is selected randomly. The sensitivity threshold is $S_{thres} = 0.05$.

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.

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 esti-mation ($\overset{|И\,|}{S}$ 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` substi-tutes 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.

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, 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.

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.

2) 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 [2] are equally likely to be leaked. 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 [3] intercepted keystroke values on a user's host. It con-nects 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 disclo-sure rate. The subset of the sensitive fingerprints is selected randomly. The sensitivity threshold is $S_{thres} = 0.05$.

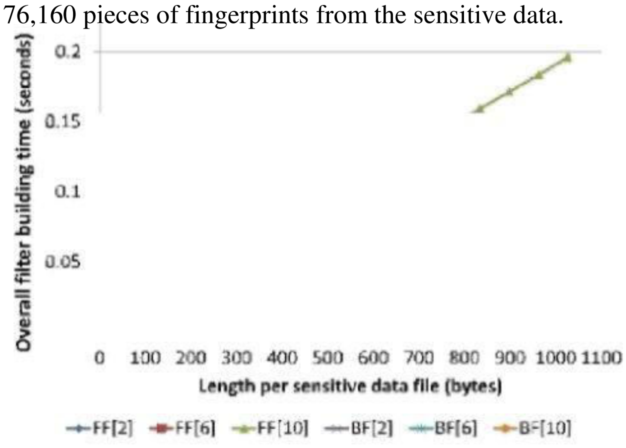
Fig. 2 shows the comparison of performance across various size of fingerprints used in the detection, in terms of the averaged sensitivity per packet in (a) and the number of detected sensitive packets in (b). These accuracy values reflect results of the POSTPROCESS operation.

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 esti-mation (S 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` substi-tutes 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.

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, 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.



(b)

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 construc-tion 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.

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.

1) 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.

VII. RELATED WORKS

In existing paper designed, implemented, and evaluated with fuzzy fingerprint technique that enhances data privacy during data-leak detection operations. It is a straightforward realization of data-leak detection requires the plaintext sensitive data. In this paper Shingle with Rabin fingerprint was used for encrypting and identifying similar spam messages in a collaborative setting. It can also able to identify spam and virus present in the message. Our proposed fuzzy fingerprint method differs from these solutions and can enable its adopter to provide data leak detection as a service. So, the data provider or customer need is not to fully trust the Data Leakage Detection provider using our proposed approach.

Besides our fingerprint-based detection, other approaches can also be applied to data-leak detection. If the sensitive data size is small and the patterns of all sensitive data are enumerable, string matching in network intrusion detection system can be used to detect data leaks. Our proposed approach can be used to detect data leaks. Another category of approaches for data-leak detection is tracing and enforcing the sensitive data flows. The provable privacy guarantees offered by our approach comes at a cost in terms of computational complexity and realization difficulty.

DISADVANTAGES

- It cost is very high
- It has high computational complexity
- It difficult in realization

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

[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 mark-ing [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

Our 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. In this paper we present SIMON-SPECK encryption algorithm for data leakage detection and identification model. By using with a special encryption algorithm, the exposure of the sensitive data is kept to a minimum during the detection. Since SIMON-SPECK algorithm uses the small key size and offers most security against multiple attacks. In this paper we listed the six operations executed by the data owner and the DLD provider in our protocol. The protocol is based on strategically computing data similarity, specifically the quantitative similarity between the sensitive information and the observed network traffic.

.

References

[1] X. Shu and D. Yao,"Privacy Preserving Detection of Sensitive Data Exposure," IEEE Transactions on Information Forensics and Security, VOL. 10, NO. 5, MAY 2015.

[2] Risk Based Security. (Feb. 2014). *Data Breach Quick-View: An Executive's Guide to 2013 Data Breach Trends.* [Online]. Available: https://www.riskbasedsecurity.com/reports/2013-DataBreachQuickView.pdf, accessed Oct. 2014.

[3] Ponemon Institute. (May 2013). *2013 Cost of Data Breach Study: Global Analysis*. [Online]. Available: https://www4.symantec.com/mktginfo/whitepaper/053013_GL_NA_WP_Ponemon-2013-Cost-of-a-Data-Breach-Report_daiNA_cta72382.pdf, accessed Oct. 2014.

[4] Identity Finder. *Discover Sensitive Data Prevent Breaches DLP Data Loss Prevention*. [Online]. Available: http://www.identityfinder.com/, accessed Oct. 2014.

[5] K. Borders and A. Prakash, "Quantifying information leaks in outbound web traffic," in *Proc. 30th IEEE Symp. Secur. Privacy*, May 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 *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 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 *Proc. 18th USENIX Secur. Symp.*, 2009, pp. 367–382.

[8] A. Nadkarni and W. Enck, "Preventing accidental data disclosure in modern operating systems," in *Proc. 20th ACM Conf. Comput. Commun. Secur.*, 2013, pp. 1029–1042.

[9] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna, "Revolver: An automated approach to the detection of evasiveweb-based malware," in *Proc. 22nd USENIX Secur. Symp.*, 2013, pp. 637–652.

[10] X. Jiang, X. Wang, and D. Xu, "Stealthy malware detection and monitor-ing through VMM-based 'out-of-the-box' semantic view reconstruction," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 2, 2010, p. 12.

[11] G. Karjoth and M. Schunter, "A privacy policy model for enterprises," in *Proc. 15th IEEE Comput. Secur. Found. Workshop*, Jun. 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 *Proc. 15th ACM Conf. Comput. Commun. Secur.*, 2008, pp. 279–288.

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

[14] K. Xu, D. Yao, Q. Ma, and A. Crowell, "Detecting infection onset with behavior-based policies," in *Proc. 5th Int. Conf. Netw. Syst. Secur.*, Sep. 2011, pp. 57–64.

[15] M. O. Rabin, "Fingerprinting by random polynomials," Dept. Math., Hebrew Univ. Jerusalem, Jerusalem, Israel, Tech. Rep. TR-15-81, 1981.

[16] A. Z. Broder, "Some applications of Rabin's fingerprinting method," in *Sequences II*. New York, NY, USA: Springer-Verlag, 1993, pp. 143–152.

[17] A. Z. Broder, "Identifying and filtering near-duplicate documents," in *Proc. 11th Annu. Symp. Combinat. Pattern Matching*, 2000, pp. 1–10.

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

[19] G. Aggarwal *et al.*, "Anonymizing tables," in *Proc. 10th Int. Conf. Database Theory*, 2005, pp. 246–258.

[20] R. Chen, B. C. M. Fung, N. Mohammed, B. C. Desai, and K. Wang, "Privacy-preserving trajectory data publishing by local suppression," *Inf. Sci.*, vol. 231, pp. 83–97, May 2013.

[21] M. O. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep. MIT/LCS/TR-212, 1979.

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

[23] W. N. Francis and H. Kucera, "Brown corpus manual," 1979.

[24] J. Kleinberg, C. H. Papadimitriou, and P. Raghavan, "On the value of private information," in *Proc. 8th Conf. Theoretical Aspects Rationality Knowl.*, 2001, pp. 249–257.

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

[26] K. Li, Z. Zhong, and L. Ramaswamy, "Privacy-aware collaborative spam filtering," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 5, pp. 725–739, May 2009.

[27] 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.

[28] F. Hao, M. Kodialam, T. V. Lakshman, and H. Zhang, "Fast payload-based flow estimation for traffic monitoring and network security," in *Proc. ACM Symp. Archit. Netw. Commun. Syst.*, Oct. 2005, pp. 211–220.

[29] L. Ramaswamy, A. Iyengar, L. Liu, and F. Douglis, "Automatic detection of fragments in dynamically generated web pages," in *Proc. 13th Int. Conf. World Wide Web*, 2004, pp. 443–454.

[30] Symantec. *Data Loss Prevention (DLP) Software*. [Online]. Available: http://www.symantec.com/data-loss-prevention/, accessed Oct. 2014.