Rapid and Parallel Content Screening for Detecting Transformed Data Exposure

Xiaokui Shu, Jing Zhang, Danfeng (Daphne) Yao, Wu-Chun Feng
Department of Computer Science, Virginia Tech
Blacksburg, VA 24060
E-mail: {subx, zjing14, danfeng, feng}@cs.vt.edu

***Abstract*—The leak of sensitive data on computer systems poses a serious threat to organizational security. Organizations need to identify the exposure of sensitive data by screening the content in storage and transmission, i.e., to detect sensitive information being stored or transmitted in the clear. However, detecting the exposure of sensitive information is challenging due to data transformation in the content. Transformations (such as insertion, deletion) result in highly unpredictable leak patterns. Existing automata-based string matching algorithms are impractical for detecting transformed data leaks because of its formidable complexity when modeling the required regular expressions. We design two new algorithms for detecting long and inexact data leaks. Our system achieves high detection accuracy in recognizing transformed leaks compared with the state-of-the-art inspection methods. We parallelize our prototype on graphics processing unit and demonstrate the strong scalability of our data leak detection solution analyzing big data.**

I. Introduction

The number of leaked records on personal computers and organization networks increases dramatically in the last years from 95 million in 2010 to 822 million in 2013 [1]. A typical approach to minimize the exposure of sensitive data is to identify all occurrences of cleartext sensitive data in storages or communications. The detection system alerts administrators of any sensitive data exposure discovered in file systems or supervised network channels. Leaks can then be identified according to the sensitive data storage and sharing policy. Different from pattern matching techniques employed in anti-virus and intrusion detection systems, data leak detection imposes new security requirements and algorithmic challenges:

1) *Data transformation.* The exposed data in the content may be transformed or modified by users or applications, so it may no longer be identical to the original sensitive data, e.g., insertions of metadata or formatting tags, substitutions of characters, and data truncation. Thus, the detection algorithm needs to recognize variations of sensitive data patterns.
2) *Scalability.* The heavy workload of data leak screening is due to *long sensitive data patterns* and the *large amount of content*. Sensitive data (e.g., customer information, documents, source code) can be of arbitrary length (e.g., megabytes). Some types of contents may need to be scanned in a timely manner (e.g., traffic scanning).

Automata-based string matching are widely used in anti-virus and network intrusion detection systems (NIDS) where the patterns to search for are static or can be characterized by regular expressions [2]. Automata are not designed to support unpredictable and arbitrary pattern variations. In data leak detection scenarios, the transformation of leaked data (in the description of regular expression) is unknown to the detection method. Creating comprehensive automata models covering all possible variations of a pattern is infeasible, because of the unrealistic high complexity. Therefore, automata approach cannot be used for detecting long and transformed data leaks.

Existing data leak detection approaches are largely based on set intersection [3]–[8]. Set intersection performed between the content fragment set and sensitive data fragment set tells the amount of sensitive data fragments appearing in the content[1]. However, set intersection is *orderless*, i.e., the order information of fragments is abandoned. Thus, set-based detection generates undesirable false alerts. In addition, set intersection cannot effectively measure the likelihood of a leak when partial data is leaked. Therefore, none of the existing techniques is adequate for detecting transformed data leaks.

The key of our solution to the detection of transformed data leaks is a new sequence alignment algorithm. *The alignment is between the sampled sensitive data sequence and the sampled content being inspected.* The alignment produces a score indicating the amount of sensitive data contained in the content. Our alignment-based solution measures the order of n-grams. It also handles arbitrary variations of patterns without an explicit specification of all possible variation patterns.

In order to deal with the scalability issue, we design a pair of algorithms to perform alignment. Our solution consists of a *comparable* sampling algorithm and a *sampling oblivious* alignment algorithm. Our sampling algorithm samples both content and sensitive data sequences. It satisfies the *comparable sampling* property that the similarity of two sequences is preserved through sampling, and the samples are meaningful to be aligned. Our solution aligns sampled sequences to infer the similarity between the original sequences before sampling. The special *sampling oblivious* property differentiates our algorithm from existing ones. Experiments show that our solution achieves accurate detection with low false positive and false negative rates. It substantially outperforms the set intersection method in terms of detection accuracy.

We design the pair of algorithms to be efficiently parallelized. We parallelize our prototype on a GPU, which achieves

[1]Typical units in a set are n-grams of a string, which preserves local features of a string and tolerates discrepancies.

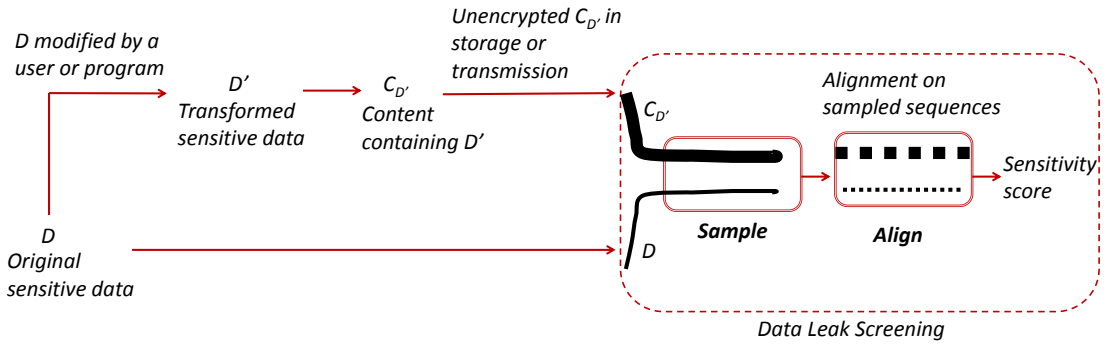

Fig. 1. A schematic drawing showing the two types of sequences in our model, their relations, and the workflow of our detection.

nearly 50 times of speedup over the simple CPU version. Our prototype reaches 400Mbps analysis throughput. This performance potentially supports the rapid security scanning of storage and communication required by a sizable organization.

II. Models and Overview

In our data leak detection model, we analyze two types of sequences: sensitive data sequence and content sequence.

- *Content sequence* is the sequence to be examined for leaks. The content may be data extracted from file systems on personal computers, workstations, and servers; or payloads extracted from supervised network channels[2].
- *Sensitive data sequence* is the information (customers' records, proprietary documents, etc.) that needs to be protected and cannot be exposed to unauthorized parties.

Both the content and the sensitive data sequences are known to the analysis system. A data leak is detected when the detection system finds a piece of sensitive data in the content sequence (Fig. 1), but the appearance is not allowed in the sensitive data storage and sharing policy. We assume that the analysis system is secure and trustworthy. Thus, the sensitive data sequence is secure during the data leak analysis. The two assumptions can be removed when our alignment is realized with secure multi-party computation techniques [9]. We do not aim at detecting stealthy data leaks that an attacker encrypts the sensitive data by herself before leaking it.

A. Technical Challenges

High detection specificity. In our data-leak detection model, high specificity refers to the ability to distinguish true leaks from coincidental matches, which can cause false alarms. Existing set-based detection is orderless, where the order of matched patterns (n-grams) is ignored. Orderless detection can result in false positives as shown below.

Sensitive data	`abcdefg`
3-grams of the sensitive data	`abc`, `bcd`, `cde`, `def`, `efg`
Content stream (false positive)	`...efg...cde...abc...`

Pervasive and localized modification. Sensitive data could be modified before it is leaked out. The modification can occur throughout a sequence (pervasive modification). The modification can also only affect a local region (local modification). We describe some modification examples:

- Character replacement, e.g., `WordPress` replaces every space character with a + in HTTP POST requests.
- String insertion: HTML tags inserted throughout a document for formatting or embedding objects.
- Data truncation or partial data leak, e.g., one page of a two-page sensitive document is transmitted.

B. Overview of Our Approach

Our work presents an efficient sequence comparison technique needed for analyzing a large amount of content for sensitive data exposure. A diagram illustrating our workflow is shown in Fig. 1. Our detection approach consists of a special sampling algorithm and a corresponding alignment algorithm working on preprocessed n-grams of sequences. The pair of algorithms computes a quantitative similarity score between sensitive data and content. Local alignment, as opposed to global alignment, is used to identify similar sequence segments, enabling the detection of partial data leaks.

Our workflow includes Extraction, Preprocessing, Sampling, Alignment, and Decision operations. The Extraction operation collects the content sequences. The Preprocessing operation prepares the sequences of n-grams for both the content and sensitive data. The Sampling operation generates samples from the preprocessed sensitive data and content sequences. The Alignment operation performs local alignment between the two sampled sequences to compute their similarity. Finally, the Decision operation confirms and reports leaks according to the sensitive data sharing policy.

III. Comparable Sampling

In this section, we formally define the new sampling requirement needed in data leak detection and present our solution.

Definition 1. *(Subsequence-preserving sampling) if string x is a substring of string y (denoted by $x \subseteq y$), then x' is also a substring of y' ($x' \subseteq y'$), where x' is a sampled subsequence of x, and y' is a sampled subsequence of y.*

In Definition 1, subsequence (with gaps) is a generalization of substring and allows gaps between characters, e.g., `lo-e` is a subsequence of `flower` (- indicates a gap).

[2]Such channels are widely used for advanced NIDS where MITM (man-in-the-middle) SSL sessions are employed to handle encryption.

The subsequence-preserving requirement in Definition 1 ensures the sample sequences of two identical strings are comparable no matter where the sampling starts or ends. We relax the identical relation to a similarity relation. If the sample sequences of two similar strings are comparable, we denote the sampling algorithm as a *comparable sampling*.

We present our comparable sampling algorithm. The advantage of our algorithm is its **context-aware selection**, i.e., the selection decision of an item depends on how it compares with its surrounding items according to a selection function. As a result, our sampling algorithm yields comparable samples.

Our comparable sampling algorithm takes in $\mathcal{S}$, an input list of items. Each item is an n-gram preprocessed from the original sensitive data or content. It outputs $\mathcal{T}$, a sampled list of the same length; the sampled list contains null values, which correspond to items that are not selected. The null regions in $\mathcal{T}$ can be aggregated, and $\mathcal{T}$ is turned into a *compact representation* $\mathcal{L}$. Each item in $\mathcal{L}$ contains the value of the sampled item and the length of the null region between the current sampled item and the preceding one.

$\mathcal{T}$ is initialized as an empty list, i.e., a list of null items. We use a small sliding window w on $\mathcal{S}$ and a selection function f to decide what items that appear in w should be selected into $\mathcal{T}$. The selection decision regarding an item is made based on not only the value of that item, but also the values of its neighboring items in w. Therefore, unlike a random sampling method where a selection decision is stochastic, our method satisfies the comparable sampling requirements.

In Algorithm 1, without loss of generality, we describe our sampling method with a specific selection function $f = \min(w, N)$. f takes in an array w and returns the N smallest items in w. All items (n-grams) are preprocessed with min-wise independent hashes (e.g., Rabin's fingerprint [10]) in our approach. Therefore, f *deterministically* selects items. The selection results at each sliding window position determine what items are chosen to the sampled list. The parameters N and $|w|$ determine the sampling rate. The directional collection difference operation `collectionDiff` in Algorithm 1 (lines 10 and 11) is similar to the set difference operation, except that it does not eliminate duplicates.

$\mathcal{T}$ output by Algorithm 1 takes the same space as $\mathcal{S}$ does. Null items can be aggregated, and $\mathcal{T}$ is turned into $\mathcal{L}$.

We have proved that Algorithm 1 is subsequence preserving. The proof is constructed in four cases: *i)* two strings are identical, *ii)* and *iii)* one is a prefix or suffix of the other, or *iv)* one is a substring of the other (but not a prefix or suffix). The complexity of sampling using the $\min(w, N)$ selection function is $O(n \log |w|)$ where n is the size of the input, $|w|$ is the size of the window. The sampling rate $\alpha \in [\frac{N}{|w|}, 1]$ approximates $\frac{N}{|w|}$ for random inputs. The detailed proof and discussion are omitted due to the space limit.

Algorithm 1 A subsequence-preserving sampling algorithm with min as the selection function.

Input: an array $\mathcal{S}$ of items, a size $|w|$ for a sliding window w, a selection function $f(w, N)$ that selects N smallest items from a window w, i.e., $f = \min(w, N)$
Output: a sampled array $\mathcal{T}$ with subsequence-preserving property

```
1: initialize T as an empty array of size |S|
2: w ← read(S, |w|)
3: let w.head and w.tail be indices in S corresponding to the
   higher-indexed end and lower-indexed end of w, respectively
4: collection m_c ← min(w, N)
5: while w is within the boundary of S do
6:     m_p ← m_c
7:     move w toward high index by 1
8:     m_c ← min(w, N)
9:     if m_c ≠ m_p then
10:        item e_n ← collectionDiff(m_c, m_p)
11:        item e_o ← collectionDiff(m_p, m_c)
12:        if e_n < e_o then
13:            write value e_n to T at w.head's position
14:        else
15:            write value e_o to T at w.tail's position
16:        end if
17:    end if
18: end while
```

IV. Sampling Oblivious Alignment

In this section, we describe the requirements for a sample-based alignment algorithm and present our solution.

We design a specialized alignment algorithm that runs on compact sampled sequences $\mathcal{L}^a$ and $\mathcal{L}^b$ to infer the similarity between the original sensitive data sequence $\mathcal{S}^a$ and the original content sequence $\mathcal{S}^b$. It needs to satisfy a new requirement **sampling oblivion**, i.e., the result of an alignment on sampled sequences $\mathcal{L}^a$ and $\mathcal{L}^b$ should be consistent with the alignment result on the original $\mathcal{S}^a$ and $\mathcal{S}^b$.

Regular local alignment without the sampling oblivion property may give inaccurate alignment on sampled sequences. However, because values of unselected items are unknown to the alignment, the decision of match or mismatch cannot be made solely on them during the alignment. We observe that leaked data region is usually consecutive, e.g., spans at least dozens of bytes. Thus, our algorithm infers the comparison outcomes between null regions based on *i)* the comparison outcomes between items surrounding null regions and *ii)* sizes of null regions. For example, given two sampled sequences `a--b` and `A--B`, if `a == A` and `b == B`, then the two values in the positions of the null regions are likely to match.

We develop our alignment algorithm with dynamic programming. A string alignment problem is divided into three prefix alignment subproblems: the current two items (from two sequences) are aligned with each other, or one of them is aligned with a gap. In our algorithm, comparison outcomes between null regions are inferred based on their non-null neighboring values and their sizes/lengths. The comparison results include *match*, *mismatch* and *gap*, and they are rewarded (match) or penalized (mismatch or gap) differently for sampled items or null regions according to a weight function $f_w()$.

We present the recurrence relation of our dynamic program alignment algorithm in Algorithm 2. For the i-th item $\mathcal{L}_i$ in a sampled sequence $\mathcal{L}$ (the compact form), the field $\mathcal{L}_i.value$ denotes the value of the item and a new field $\mathcal{L}_i.span$ denotes the size of null region between that item and the preceding non-null item. Our alignment algorithm computes a non-negative score matrix H of size $|\mathcal{L}^a|$-by-$|\mathcal{L}^b|$ for the input

Algorithm 2 Recurrence relation in dynamic programming.

Input: A weight function f_w, visited cells in H matrix that are adjacent to $H(i,j)$: $H(i-1,j-1)$, $H(i,j-1)$, and $H(i-1,j)$, and the i-th and j-th items $\mathcal{L}_i^a, \mathcal{L}_j^b$ in two sampled sequences $\mathcal{L}^a$ and $\mathcal{L}^b$, respectively.

Output: $H(i,j)$

1: $h^{up}.score \leftarrow f_w(\mathcal{L}_i^a, -, H(i-1,j))$

2: $h^{left}.score \leftarrow f_w(-, \mathcal{L}_j^b, H(i,j-1))$

3: $h^{dia}.score \leftarrow f_w(\mathcal{L}_i^a, \mathcal{L}_j^b, H(i-1,j-1))$

4: $h^{dia}.null_{row} \leftarrow \begin{cases} 0, & \text{if } \mathcal{L}_i^a = \mathcal{L}_j^b \\ H(i-1,j).null_{row} + \mathcal{L}_i^a.span + 1, & \text{else} \end{cases}$

5: $h^{dia}.null_{col} \leftarrow \begin{cases} 0, & \text{if } \mathcal{L}_i^a = \mathcal{L}_j^b \\ H(i,j-1).null_{col} + \mathcal{L}_j^b.span + 1, & \text{else} \end{cases}$

6: $H(i,j) \leftarrow \arg\max_{h.score} \left\{h^{up}, h^{left}, h^{dia}\right\}$

7: $H(i,j).score \leftarrow \max\{0, H(i,j).score\}$

sequence $\mathcal{L}^a$ and $\mathcal{L}^b$ and returns the maximum alignment score with respect to a weight function. Each cell $H(i,j)$ has a score field $H(i,j).score$ and two extra fields recording sizes of neighboring null regions, namely $null_{row}$ and $null_{col}$. $null_{row}$ and $null_{col}$ fields for all three cell candidates h^{up}, h^{left}, h^{dia} are initialized as 0.

Our weight function $f_w()$ takes three inputs: the two items being aligned (e.g., $\mathcal{L}_i^a$ from sensitive data sequence and $\mathcal{L}_j^b$ from content sequence) and a reference cell c (one of the three visited adjacent cells $H(i-1,j-1)$, $H(i,j-1)$, or $H(i-1,j)$), and outputs a score of an alignment configuration. One of $\mathcal{L}_i^a$ and $\mathcal{L}_j^b$ may be a gap ($-$) in the alignment. The computation is based on the penalty given to mismatch and gap conditions and reward given to match condition. $f_w()$ in Algorithm 2 differs from the typical weight function in Smith-Waterman algorithm [11] in its ability to infer comparison outcomes for null regions. This inference is done accordingly to the values of their adjacent non-null neighboring items. The details of $f_w()$ are omitted due to the space limit.

The complexity of Algorithm 2 is $O(|\mathcal{L}^a||\mathcal{L}^b|)$, where $|\mathcal{L}^a|$ and $|\mathcal{L}^b|$ are lengths of compact representations of the two sampled sequences. If the length of the sensitive data is fixed, the complexity deduces to $O(|\mathcal{L}^b|)$. Our alignment over two sampled sequences achieves a speedup in the order of $O(\alpha^2)$, where $\alpha \in (0,1)$ is the sampling rate. Details of the analysis are omitted due to the space limit.

V. Evaluation on Detection Accuracy

We evaluate the accuracy of our solution with several large datasets under real-world data leak scenarios[3]. We implement a single-threaded prototype (referred to as *AlignDLD* system) and a collection intersection method as a baseline. Both systems are written in `C++`, compiled using `g++` 4.7.1 with flag `-O3`. We also provide two parallel versions of our prototype in Section VI for performance demonstration.

- *AlignDLD:* our sample-and-align data leak detection method with sampling parameters $N = 10$ and $|w| = 100$. 3-grams and 32-bit Rabin's fingerprints[4] are used.
- *Coll-Inter:* a data leak detection system based on collection intersection[5], which is widely adopted by commercial tools such as GlobalVelocity and GoCloudDLP. 8-grams and 64-bit Rabin's fingerprints are used, which is standard with collection intersection.

A. Experiment Setup

We reproduce four leaking scenarios in a virtual network environment using VirtualBox. The network traffic content is monitored at the virtual gateway of the virtual network.

1) *Web leak:* a user publishes sensitive data on the Internet via typical publishing services, e.g., `WordPress`,
2) *FTP:* a user transfers unencrypted sensitive files to an FTP server on the Internet,
3) *Backdoor:* a malicious program, i.e., `Glacier`, on the user's machine exfiltrates sensitive data,
4) *Spyware:* a `Firefox` extension `FFsniFF` [12] exfiltrates sensitive information via web forms.

We test several datasets listed below. The first two are used for both sensitive data and content sequences. The last two are only used as content. Due to the space limit, we only present two accuracy experiments using *A. Enron* dataset in this paper.

A. *Enron dataset* (2.6GB) consists of 517,424 real emails from 150 users [13],
B. *Source-code* includes 288 source code files from `gzip`, `procps`, `net-tools`, `tar`, and `rsync` projects,
C. *HTTP headers* (12MB) contains critical URL information in HTTP requests. We collect the data from 20 users' Internet surfing (30 minutes for each user),
D. *MiscNet* (500MB) is an Internet traffic dump containing various kinds of Internet traffic, e.g., web surfing, video, email, instant messaging, downloading, etc.

For evaluation purposes, we explain our accuracy measures. If a real leak is detected, it is a true positive. If a non-leak traffic content is detected as a leak, it is a false positive. Detection rate or recall is $\frac{\text{TP}}{\text{TP + FN}}$. False positive rate is $\frac{\text{FP}}{\text{FP + TP}}$.

We define the *sensitivity* $\mathbb{S} \in [0,1]$ of the content sequence in Formula 1. It indicates the similarity of sensitive data D and content $C_{D'}$ with respect to their sequences $\mathcal{S}^a$ and $\mathcal{S}^b$ after PREPROCESS. ξ is the maximum score in the alignment, and r is the reward for one-unit match in the alignment.

$$\mathbb{S} = \frac{\xi}{r \times \min\left(|\mathcal{S}^a|, |\mathcal{S}^b|\right)} \tag{1}$$

B. Detecting Modified Leaks

We present the detection results against modified leaks in this subsection. We run AlignDLD and Coll-Inter on three types of data leaks listed below.

1) Content without any leak, i.e., the content does not contain any sensitive data.
2) Content with unmodified leak, i.e., sensitive data appearing in the content is not modified.
3) Content with modified leaks caused by `WordPress`, which substitutes every space with "+" in the content.

[3]We only present the most important experiments due to the space limit.

[4]Rabin's fingerprint is used for unbiased sampling (Section III).

[5]Set and collection intersections are used interchangeably in this paper.

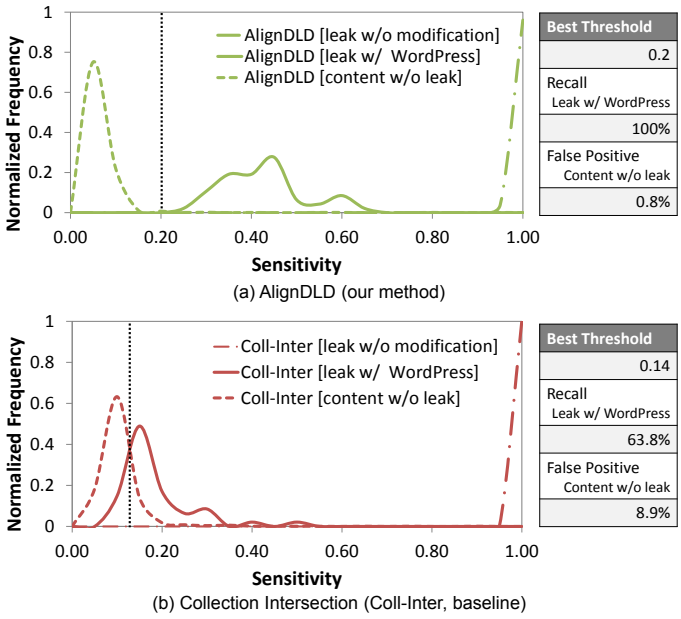

Fig. 2. Detection comparison of leak through `WordPress` using AlignDLD (a) and collection intersection (b). Solid lines show the sensitivity distribution of the modified leaks via `WordPress`.

The content and sensitive data in this experiment are selected emails from *A. Enron* dataset. The content without leak consists of 950 randomly chosen Enron emails, and the sensitive data consists of 50 randomly chosen ones. We compute the sensitivities of the content according to Equation 1.

We evaluate and compare AlignDLD and Coll-Inter. We present the distributions of all sensitivity values in Fig. 2. The table to the right of each figure summarizes the detection accuracy under the best threshold. Both methods perform as expected in the scenarios of no-leak (dotted lines on the left) and unmodified leak (dashed lines on the right).

The solid lines in Figure 2 represent the detection results of leaks with `WordPress` modifications. Our AlignDLD method in Fig. 2 (a) gives much higher sensitivity scores to the transformed data leak than the Coll-Inter method. With a threshold of 0.2, **all the email messages with transformed leaks are detected,** i.e., it achieves 100% recall. The false positive rate is low. In contrast, the collection intersection method in Fig. 2 (b) has a significant overlap between messages with no leak and messages with transformed leaks. Its accuracy is much lower than that of our method, e.g., 63.8% recall and a 10 times higher false positive rate. Further analysis on false positives caused by coincidental matches (dotted lines on the left) is omitted due to the space limit.

C. Partial Data Leaks

In data truncation or partial data leak scenarios, consecutive portions of the sensitive data are leaked. In this experiment, a content sequence contains a portion of sensitive text. The total length of the sensitive text is 1KB. The size of the leaked portion in the content ranges from 32 bytes to 1KB. Each content sequence is 1KB long with random padding.

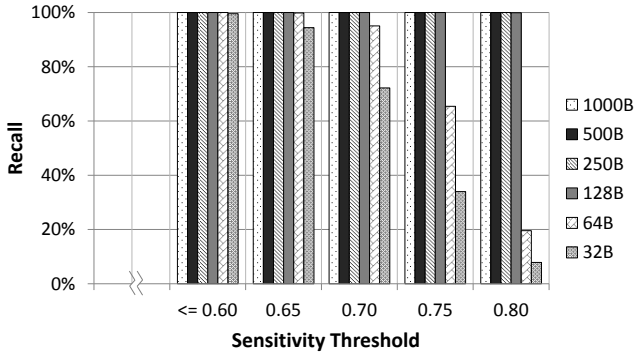

Fig. 3. The detection success rate of AlignDLD in partial data leaks under various detection thresholds.

We measure the *unit sensitivity* $\tilde{\mathbb{S}} \in [0,1]$ on segments of content sequences. Unit sensitivity $\tilde{\mathbb{S}}$ is the normalized *per-element* sensitivity value for the aligned portion of two sequences. It is defined in Equation 2, where $\tilde{\xi}$ is the maximum local alignment score obtained between aligned segments $\tilde{\mathcal{S}}^a$ and $\tilde{\mathcal{S}}^b$, which are sequence segments of sensitive data D and content $C_{D'}$. The higher $\tilde{\mathbb{S}}$ is, the better the detection is. Threshold l is a predefined length describing the shortest segment to invoke the measure. $l = 16$ in our experiments.

$$\tilde{\mathbb{S}} = \frac{\tilde{\xi}}{r \times \min\left(|\tilde{\mathcal{S}}^a|, |\tilde{\mathcal{S}}^b|\right)} \quad \text{where} \quad \min\left(|\tilde{\mathcal{S}}^a|, |\tilde{\mathcal{S}}^b|\right) \geq l \qquad (2)$$

The detection results of AlignDLD are shown in Figure 3. Content with longer sensitive text is easier to detection as expected. Nevertheless, **our method detects content with short truncated leaks as small as 32 bytes with high accuracy.** The detection rate decreases with higher thresholds. We observe that high thresholds (e.g., > 0.6) are not necessary for detection when 8-byte shingles are used; false positives caused by coincidental matches are low in this setup. These experiments show that our detection is resilient to partial data leaks or data truncation.

VI. Parallelization and Evaluation

In order to achieve high analysis throughput, we parallelize our algorithms on CPU as well as on general-purpose GPU platforms. The multithreading CPU version and the GPU versions are implemented to demonstrate the strong scalability[6] and performance potential of our algorithms.

We implement a multithreading CPU version and a GPU version [7] of our prototype on a hybrid CPU-GPU machine equipped with an Intel Core i5 2400 and an NVIDIA Tesla C2050 GPU (Fermi architecture with 448 GPU cores).

In the multithreading CPU version, we parallelize both the Sampling and Alignment procedures with the `pthread` library. Long streams are split into multiple substrings, which are sampled in parallel by different threads and then assembled

[6]The scalability experiments are not shown due to the space limit.

[7]The multithreaded CPU version is written in C, compiled using gcc 4.4.5 with flag -O2. The GPU version is written in CUDA compiled using CUDA 4.2 with flag -O2 -arch sm 20 and NVIDIA driver v295.41.

TABLE I
TIMES OF SPEEDUP IN ALIGNDLD'S ALIGNMENT OPERATION. [P] REPRESENTS A PARALLEL VERSION.

Traffic	*A.Enron*			*D.MiscNet*		
Data	txt	png	pdf	txt	png	pdf
CPU	1.00	1.00	1.00	1.00	1.00	1.00
CPU[P]	3.59	3.29	3.40	2.78	3.18	2.82
GPU[P]	44.36	47.93	47.98	34.60	42.51	41.59

TABLE II
THROUGHPUT (IN MBPS) OF THE ALIGNMENT OPERATION ON GPU

Sensitive data size (KB)	250	500	1000	2500
Sampling rate				
0.03	426	218	110	44
0.12	23	11	5	2

for output. ALIGNMENT is the most time-consuming procedure and is made parallel on both CPU and GPU. We use a parallelized score matrix filling method to compute a diagonal of cells at the same time. This method consumes linear space.

We evaluate the performance of the most time-consuming ALIGNMENT procedure on the Tesla GPU. Times of speedup in detecting sensitive data of types `txt`, `png`, or `pdf` against *A. Enron* or *D. MiscNet* traffic, respectively, are shown in Table I. Our GPU detection prototype achieves over 40 times of speedup over the CPU version on large content datasets. The prototype achieves a throughput of over 400Mbps against 500MB misc network traffic (*D. MiscNet*) shown in Table II. This throughput is comparable to that of a moderate commercial firewall. More optimizations on data locality and memory usage can be performed in real-world detection products.

VII. RELATED WORK

Existing commercial tools and services for data leak prevention include Symantec DLP [5], IdentityFinder [6], GlobalVelocity [7], and GoCloudDLP [8]. All solutions are likely based on set intersection. Symantec DLP is based on n-grams and Bloom filters that save space for set membership test.

Network intrusion detection systems (NIDS) such as Snort and Bro use regular expression to perform string matching in deep packet inspection [14]. Nondeterministic finite automaton (NFA) with backtracking requires $O(2^n)$ time and $O(n)$ space. Deterministic finite automaton (DFA) has a time complexity of $O(n)$ and a space complexity of $O(2^n)$. Neither DFA or NFA is designed to support arbitrary and unpredictable pattern variations. In comparison, our sequence alignment solution covers all possible pattern variations in long sensitive data without explicitly specifying them.

Alignment algorithms have been widely used in computational biology applications [15]. Jung et al. proposed NetDialign for network privacy using the Dialign algorithm [16]. Kreibich and Crowcroft presented an alignment algorithm for traffic intrusion detection [17]. Data leak detection differs from the above network privacy and IDS problems. We consider both transformed leaks and scalability issue. Our alignment performs complex inferences needed for aligning sampled sequences, and our solution is also different from fast non-sample alignment in bioinformatics, e.g., BLAST [18].

Other approaches to detect and prevent data leaks include tracing data movement within the device [19], searching short sensitive keywords [20], symbolic execution analysis on mobile apps [21], etc. These methods differ from ours because we aim at comparing long and inexact sequences from sensitive data and content sequences.

VIII. CONCLUSIONS

Despite the commercial success of data leak software and appliances, existing solutions based on set intersection have serious security drawbacks. We present new and sophisticated alignment-based algorithms to improve the accuracy detecting data leaks, e.g., high specificity (i.e., low false alarm rate). Our extensive experimental evaluations with real-world data and leak scenarios confirm that our method is useful for big data analytics and detecting transformed sensitive data exposure.

REFERENCES

[1] RiskBasedSecurity, "Data breach quickview: An executive's guide to 2013 data breach trends," February 2014.

[2] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," *Commun. ACM*, vol. 18, no. 6, pp. 333–340, 1975.

[3] X. Shu and D. Yao, "Data leak detection as a service," in *Proceedings of SecureComm*, September 2012, pp. 222–240.

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

[5] Symantec, "Symantec data loss prevention," http://www.symantec.com/data-loss-prevention.

[6] identifyfinder, "Identity finder," http://www.identityfinder.com/.

[7] Global Velocity Inc., "Global velocity," http://www.globalvelocity.com/.

[8] GTB Technologies Inc., "GoCloudDLP," http://www.goclouddlp.com/.

[9] S. Jha, L. Kruger, and V. Shmatikov, "Towards practical privacy for genomic computation," in *Proceedings of IEEE S&P*, 2008.

[10] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *J. Comput. Syst. Sci.*, vol. 60, no. 3, pp. 630–659, 2000.

[11] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J. Mol. Biol.*, vol. 147, no. 1, pp. 195–197, March 1981.

[12] C. Wuest and E. Florio, "Firefox and malware: When browsers attack," Symantec Corporation, Tech. Rep., October 2009.

[13] C. Kalyan and K. Chandrasekaran, "Information leak detection in financial e-mails using mail pattern analysis under partial information," in *Proceedings of the 7th WSEAS International Conference on Applied Informatics and Communications*, vol. 7, 2007, pp. 104–109.

[14] P.-C. Lin, Y.-D. Lin, Y.-C. Lai, and T.-H. Lee, "Using string matching for deep packet inspection," *IEEE Computer*, vol. 41, pp. 23–28, 2008.

[15] V. O. Polyanovsky, M. A. Roytberg, and V. G. Tumanyan, "Comparative analysis of the quality of a global algorithm and a local algorithm for alignment of two sequences," *Algorithms Mol Biol*, vol. 6, p. 25, 2011.

[16] 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 *ACM CCS*, 2008, pp. 279–288.

[17] C. Kreibich and J. Crowcroft, "Efficient sequence alignment of network traffic," in *Proceedings of IMC*, 2006, pp. 307–312.

[18] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.

[19] A. Nadkarni and W. Enck, "Preventing accidental data disclosure in modern operating systems," in *Proceedings of ACM CCS*, 2013.

[20] V. P. Kemerlis, V. Pappas, G. Portokalidis, and A. D. Keromytis, "iLeak: A lightweight system for detecting inadvertent information leaks," in *Proceedings of EC2ND*, October 2010.

[21] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "AppIntent: Analyzing sensitive data transmission in Android for privacy leakage detection," in *Proceedings of ACM CCS*, 2013.