Chapter 5
Rapid Screening of Big Data Against Inadvertent Leaks

Xiaokui Shu, Fang Liu, and Danfeng (Daphne) Yao

Abstract Keeping sensitive data from unauthorized parties in the highly connected world is challenging. Statistics from security firms, research institutions, and government organizations show that the number of data-leak instances has grown rapidly in the last years. Deliberately planned attacks, inadvertent leaks, and human mistakes constitute the majority of the incidents. In this chapter, we first introduce the threat of data leak and overview traditional solutions in detecting and preventing sensitive data from leaking. Then we point out new challenges in the era of big data and present the state-of-the-art data-leak detection designs and algorithms. These solutions leverage big data theories and platforms—data mining, MapReduce, GPGPU, etc.—to harden the privacy control for big data. We also discuss the open research problems in data-leak detection and prevention.

5.1 Introduction: Data Leaks in the Era of Big Data

The exposure of sensitive data has become a severe threat to organizational and personal security. For a period of 5 years from 2010 to 2014, RiskBasedSecurity, a cyber security firm, reported a ten times growth of leaked records. These leaked records include credit card numbers, medical records, classified documents, etc. The total number of leaked records in 2014 reached a record high of 1.1 billion through 3014 incidents [53]. Kaspersky Lab estimated the average damage cost by an incident to be $720,000 [35], and FBI warned retailers about the increasing threat of data leaks in 2014 [20].

To understand data leaks and find defenses against the threat, we classify data-leak incidents into two categories by their causes:

- *Intentional leaks* include both external and internal threats such as network intrusion, phishing, espionage, virus, etc.
- *Inadvertent leaks* include accidental data sharing by employees, transmitting confidential data without proper encryption, etc.

X. Shu (✉) • F. Liu • D. (Daphne) Yao
Department of Computer Science, Virginia Tech, Blacksburg, VA 24060, USA
e-mail: subx@cs.vt.edu; fbeyond@cs.vt.edu; danfeng@cs.vt.edu

S. Yu, S. Guo (eds.), *Big Data Concepts, Theories, and Applications*,
DOI 10.1007/978-3-319-27763-9_5

Recent intentional data-leak incidents include SONY Picture, Home Depot, Target, Neiman Marcus, P.F. Chang, and Michaels and Aaron Brother. They hit the headlines of newspapers and Internet media for their data breaches during the last 2 years (2013–2014). For instance, the Target Corporation's network was breached between November 27 and December 18, 2013. 40 million credit and debit card numbers and 70 million records of personal information were stolen in this single incident. Months later, this number was surpassed by the Home Depot breach and the SONY Picture breach.

Many inadvertent leaks, e.g., forwarding a confidential email outside the company, are widely seen in companies and organizations. Such incidents can be found in 29 % of all 4438 data leaks reported by Kaspersky Lab during a 12-month period from 2013 to 2014 [35]. Although inadvertent leaks do not result in explicit economic loss of a company, they can be further exploited to launch effective attacks through these implicit leaking channels.

Encrypting data in the storage is one of the most basic methods preventing sensitive data from leaking, and it is effective against external leaks if used properly. It nullifies data leaks by preserving data in encrypted forms even when the data is stolen. However, encryption does not prevent sensitive data from leaking when the data is decrypted and consumed by processes in memory. In this case, software vulnerabilities and other internal threats can still cause data leaks even they are properly encrypted.

Besides general security mechanisms that enforce organizational security and privacy to prevent data leaks, specific solutions countering data leaks can be deployed to detect data-leak events and perform countermeasures against the events. Data-leak detection (DLD) is a solution that reveals data-leak events in a single device or among a network of devices. The detection usually combines a multitude of techniques that target different causes of data leaks.

Traditional virus detection, firewalls, and intrusion detection are basic elements to detect and prevent intentional leaks. A proper combination of these basic systems provides a comprehensive detection against sophisticated attack vectors, such as advanced persistent threats (APT), which are responsible for many significant data breach incidents.

Data tracking [15, 26, 44, 69] is an approach against both intentional and inadvertent data leaks. The basic idea is to track the flow of sensitive data within a single device or a network so that unauthorized data flows are prohibited or stopped. The detection tracks every segment of memory that the sensitive data is stored. Data tracking approach may incur heavy overhead at runtime. In addition, it may require a network-wide memory tracking solution across distributed systems/devices.

Content screening is an approach to detect data leaks at critical sites in a device (e.g., a network interface) or in a network (e.g., a gateway), but not to track the flow of sensitive data anywhere, anytime. It inspects data flows at critical sites (e.g., data flow boundaries) and recognizes sensitive data in the content if any. It is practical that detection-related computations are only performed at specific sites in a device or a network. Several commercial data-leak detection solutions are based on the content screening approach [23, 24, 28, 62]. The key of content screening for

detection is the **recognition of sensitive data**. The detection system is deployed at critical data flow boundaries. It extracts the content from intercepted data flows and seeks the trace of sensitive data in the content. Although this strategy is practical and costs less than sensitive data tracking, it does not detect stealthy intentional data leaks that could be privately encrypted by an attacker. Therefore, this technique is mostly used for detecting inadvertent data leaks.

A basic technique to *recognize sensitive data* from data flow content is to compute the set intersection rate between the set of n-grams from the content and the set of n-grams from the sensitive data. The use of n-grams preserves local features of sensitive data and reduces false positives. The set-intersection-based detection is versatile, capable of analyzing both text and some binary-encoded context (e.g., Word or `pdf` files). The method has been used to detect similar documents on the web [8], shared malicious traffic patterns [10], malware [30], as well as email spam [40]. Set intersection based detection is simple to implement, and can be found in several commercial products for data-leak detection [23, 24, 28, 62].

However, the era of big data brings new challenges for data-leak detection.

- *Scalability Challenge* A naive implementation of the set intersection procedure requires $O(nm)$ complexity, where n and m are sizes of the two sets A and B, respectively. If the sets are relatively small, then a faster implementation is to use a hashtable to store set A and then testing whether items in B exist in the hashtable or not, giving $O(n + m)$ complexity. However, if A and B are both very large, a naive hashtable may have hash collisions that slow down the computation. Increasing the size of the hashtable may not be practical due to memory limitation and thrashing.
- *Accuracy Challenges* As the size of sensitive data increases, the accuracy of the detection is heavily affected by two accuracy challenges.
  - *Transformed Data Leaks.* The exposed data in the content may be unpredictably transformed or modified by users or applications, and it may no longer be identical to the original sensitive data, e.g., insertions of metadata or formatting tags, substitutions of characters for formatting purposes. It reduces the accuracy of set-intersection-based approaches.
  - *Partial Data Leaks.* The exposed data in the content may be a consecutive segment of sensitive documents instead of entire pieces of sensitive data. It is an extreme case of deletion (one kind of transformation in general) where most of sensitive data is removed. It is listed as a separate challenge since it completely nullifies set-intersection-based approaches.
- *Privacy Challenge* Cloud computing is one of the key infrastructures in the era of big data, which enables the storage and processing of large volumes of data. However, conventional set intersection operations require the possession of the sensitive data. The requirement makes it improper to outsource data-leak detection procedures to a third party, i.e., the DLD provider. Simply hashing sensitive n-grams does not solve the problem, because the DLD provider can learn the sensitive data from data flows that contain data leaks.

In this chapter, we describe two detection solutions in details that are specifically designed to address the big data challenges for data-leak detection.

5.1.1 MR-DLD: Privacy-Preserving Data-Leak Detection Through MapReduce Collection Intersection

MR-DLD leverages MapReduce [17], a programming model for distributed data-intensive applications, to realize collection intersection operations in parallel and perform data-leak detection. The detection is distributed and parallel, capable of screening massive amount of content for exposed information.

The advantage of MR-DLD is its scalability and privacy-preserving features. Because of the intrinsic $\langle key, value \rangle$ organization of items in MapReduce, the worst-case complexity of MR-DLD is correlated with the size of the leak (specifically a $\gamma \in [0, 1]$ factor denoting the size of the intersection between the content collection and the sensitive data collection). This complexity reduction brought by the γ factor is significant because the value is extremely low for normal content without a leak. In MR-DLD, items not in the intersection (non-sensitive content) are quickly dropped without further processing. Therefore, the MapReduce-based algorithms have a lower computational complexity compared to the single-host collection-intersection implementation.

The data privacy protection is realized using fast one-way transformation. This transformation requires the pre- and post-processing by the data owner for hiding and precisely identifying the matched items, respectively. Both the sensitive data and the content need to be transformed and protected by the data owner before it is given to the MapReduce nodes for the detection. In the meantime, such a transformation has to support the equality comparison required by the collection intersection. In addition, the one-way transformation is updated with new key for each detection session. The periodical updates prevent an adversary from learning frequency information from content digests and performing frequency analysis. This technique provides strong privacy guarantee for the data owner, in terms of the low probability for a MapReduce node to recover the sensitive data.

5.1.2 AlignDLD: Data-Leak Detection Through Alignment

AlignDLD solves the challenge of transformed and partial data-leak detection through a specially designed sequence alignment algorithm. *The alignment is between the sampled sensitive data sequence and the sampled content being inspected.* The alignment produces scores indicating the amount of sensitive data contained in the content.

The advantage of AlignDLD is its accuracy and scalability. AlignDLD measures the order of *n*-grams. It also handles arbitrary variations of patterns without an explicit specification of all possible variation patterns. Experiments show that AlignDLD substantially outperforms the collection-intersection-based methods in terms of detection accuracy in a multitude of transformed data-leak scenarios.

The scalability issue is solved in AlignDLD by sampling both the sensitive data and content sequences before aligning them. This procedure is enabled by a *comparable* sampling algorithm and a *sampling-oblivious* alignment algorithm in a pair. The comparable sampling algorithm yields constant samples of a sequence wherever the sampling starts and ends. The sampling-oblivious alignment algorithm infers the similarity between the original unsampled sequences with sophisticated traceback techniques through dynamic programming. The algorithm infers the lost information (i.e., sampled-out elements) based on the matching results of their neighboring elements. Evaluation results show that AlignDLD boosts the performance, yet only incurs a very small amount of mismatches.

The rest of the chapter is organized as follows. We first formalize the basic set intersection model for conventional data-leak detection in Sect. 5.2. In the next two sections, we detail the two data-leak detection solutions solving big data challenges in Sects. 5.3 and 5.4, respectively. Then we give the literature review on data-leak detection and related techniques in Sect. 5.5. We discuss the open problems in the field and conclude the chapter in Sects. 5.6 and 5.7.

5.2 Model and Background

In a typical content-screening data-leak detection model, two types of sequences, i.e., sensitive data sequence and content sequence, are analyzed.

- *Content sequence* is the sequence to be examined for leaks. The content may be extracted from file systems on personal computers, workstations and servers, or from payloads extracted from network traffic.
- *Sensitive data sequence* contains the information (e.g., customers' records, proprietary documents) that needs to be protected and cannot be exposed to unauthorized parties. The sensitive data sequence should not be known to the analysis system if it is not secure and trustworthy (e.g., detection is performed by cloud or DLD provider).

5.2.1 Security Model

We classify data leaks into two categories according to their causes:

- Case I *Inadvertent data leak*: The sensitive data is accidentally leaked in the outbound traffic by a legitimate user. This chapter focuses on detecting this

type of accidental data leaks over supervised network channels. Inadvertent data leak may be due to human errors such as forgetting to use encryption, carelessly forwarding an internal email and attachments to outsiders, or due to application flaws (such as described in [33]). 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 [31].

- Case II *Intentional 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 the network-based solution. Host-based defenses (such as detecting the infection onset [67]) need to be deployed instead.

In this chapter, we describe data-leak detection solutions against Case I, the inadvertent data leaks over supervised network channels. In other words, the detection techniques in this chapter aim to discover sensitive data appearance in network traffic over supervised network channels. We assume that: (1) plaintext data in supervised network channels can be extracted for inspection; (2) the data owner is aware of legitimate data transfers; and (3) 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 detection system.

5.2.2 Basic Solution

The basic approach for detecting data leak is based on computing the similarity between content sequences and sensitive data sequences, specifically **the intersection of two collections of shingles**. 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, the basic approach can tolerate sensitive data modification to some extent, e.g., inserted tags, a small amount of character substitution, and lightly reformatted data. The use of shingles for finding duplicate web documents first appeared in [7, 8].

One collection consists of shingles obtained from the content sequence and the other collection consists of shingles from the sensitive sequence. Collection intersection differs from set intersection, in that it also records duplicated items in the intersection, which is illustrated in Fig. 5.1. Recording the frequencies of

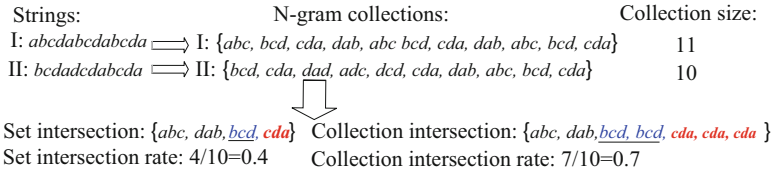

Fig. 5.1 An example illustrating the difference between set intersection and collection intersection in handling duplicates for 3-grams

intersected items achieves more fine-grained detection. Thus, collection intersection is preferred for data-leak analysis than set intersection.

Formally, given a content collection C_c and a sensitive data collection C_s, the detection algorithms aim to compute the intersection rate $Irate \in [0, 1]$ defined in (5.1), where $Inum$ is the occurrence frequency of an item i in the intersection $C_s \cap C_c$. The sum of frequencies of all items appeared in the collection intersection is normalized by the size of the sensitive data collection (assuming $|C_s| < |C_c|$), which yields the intersection rate $Irate$. The rate represents the percentage of sensitive data that appears in the content. $Irate$ is also referred to as the *sensitivity score* of a content collection.

$$Irate = \frac{\sum_{i \in \{C_s \cap C_c\}} Inum_i}{|C_s|} \tag{5.1}$$

Existing commercial products for data-leak detection/prevention are likely based on the basic solution introduced above in Sect. 5.2.2. These products include Symantec DLP [62], IdentityFinder [28], GlobalVelocity [23], and GoCloud-DLP [24]. Data-leak detection techniques used in the literature include keyword searching [36], data tracking [26, 44, 69], watermarking [2, 46], etc.

The most severe problem with these techniques in big data era is that they are not able to process massive content data in time. In addition, none of them address the privacy challenges, meaning that the person or company who performs detection is possible to learn about the sensitive data.

MapReduce has been used to address the scalability challenge in data mining [78], machine learning [45], database [4, 64], and bioinformatics [43]. It is also recently used in security areas such as log analysis [22, 68], spam filtering [12, 13] and malware detection [21, 49, 79]. Privacy-preserving techniques are also widely invented for secure multi-party computation [70]. Shingle with Rabin fingerprint [50] is also used for collaborative spam filtering [40], worm containment [10], virus scan [25], and fragment detection [52].

These techniques laid the foundation for the two new approaches below on detecting data leak in massive content.

5.3 MR-DLD: MapReduce-Based Data-Leak Detection

To address the scalability challenges, we present a data-leak detection system in MapReduce (MR-DLD) [41]. MapReduce [17] is a programming framework for distributed data-intensive applications. It has been used to solve big data security problems such as spam filtering [12, 13], Internet traffic analysis [39] and log analysis [4, 42, 68]. MapReduce algorithms can be deployed on nodes in the cloud or in local computer clusters.

The detection also provides privacy enhancement to preserve the confidentiality of sensitive data during the outsourced detection. Because of this privacy enhancement, the MapReduce algorithms can be deployed in distributed environments where the operating nodes are owned by third-party service providers. Applications of the MR-DLD system include data-leak detection in the cloud and outsourced data-leak detection.

5.3.1 Threat Model

In this model, two parties participate in the large-scale data-leak detection system: data owner and data-leak detection (DLD) provider.

- *Data owner* owns the sensitive data and wants to know whether the sensitive data is leaked. It has the full access to both the content and the sensitive data. However, it only has limited computation and storage capability and needs to authorize the DLD provider to help inspect the content for the inadvertent data leak.
- *DLD provider* provides detection service and has unlimited computation and storage power when compared with data owner. It can perform offline inspections on big data without real-time delay. However, the DLD provider is honest-but-curious (aka semi-honest). That is, it follows the prescribed protocol but may attempt to gain knowledge of sensitive data. The DLD provider is not given the access to the plaintext content. It can perform dictionary attacks on the signature of sensitive data records.

The goal of MR-DLD is to offer DLD provider the solution to scan massive content for sensitive data exposure and minimize the possibility that the DLD provider learns about the sensitive information.

5.3.2 Confidentiality of Sensitive Data

Naive collection-intersection solutions performing on *shingles* provide no protection for the sensitive data. The reason is that MapReduce nodes can easily reconstruct sensitive data from the shingles. MR-DLD utilizes several methods for

the data owner to transform shingles before they are released to the MapReduce nodes. These transformations, including specialized hash function, provide strong-yet-efficient confidentiality protection for the sensitive information. In exchange for these privacy guarantees, the data owner needs to perform additional data pre- and post-processing operations.

In addition to protecting the confidentiality of sensitive data, the pre-processing operations also need to satisfy the following requirements:

- Equality-preserving: the transformation operation should be deterministic so that two identical shingles within one session are always mapped to the same item for comparison.
- One-wayness: the function should be easy to compute given any shingle and hard to invert given the output of a sensitive shingle.
- Efficiency: the operation should be efficient and reliable so that the data owner is able to process very large content.

The collection intersection (in Sect. 5.3.4) is computed on one-way hash values of n-grams, specifically Rabin fingerprints. Rabin fingerprint is a fast one-way hash function, which is computational expensive to invert. In addition, Rabin fingerprints can be computed in linear time [7]. The computation can be implemented with fast XOR, shift and table lookup operations.

Specifically, Rabin fingerprint of a n-bit shingle is based on the coefficients of the remainder of the polynomial modular operation with an irreducible polynomial $p(x)$ as the modulo as shown in (5.2), where c_{n-i+1} is the i-th bit in the shingle C.

$$f(C) = c_1 x^{n-1} + c_2 x^{n-2} + \ldots + c_{n-1} x + c_n \; mod \; p(x) \tag{5.2}$$

To meet the privacy requirements, the MR-DLD approach expands the Rabin fingerprint and presents a new "keyed-hash" operation as shown in (5.3), where K is data owner selected secret session key and S is the input shingle.

$$f(K, p(x), S) = K \oplus (s_1 x^{n-1} + s_2 x^{n-2} + \ldots + s_{n-1} x + s_n \; mod \; p(x)) \tag{5.3}$$

The difference between the expanded operation and Rabin fingerprint is that $p(x)$ and K change periodically as parameters. Different from a regular keyed-hash method, the "key" in the MR-DLD operation is used to keep updating the fingerprint method. In this chapter, we refer to the expanded operation as Rabin fingerprint. Section 5.3.5 presents the security analysis of the MR-DLD approach especially on the confidentiality of sensitive data.

5.3.3 Technical Requirements and Design Overview

In this section, we introduce MapReduce and the specific challenges when performing data-leak detection with MapReduce. We further present the workflow of the MR-DLD detection framework and the overview of the collection intersection algorithm used in the framework.

5.3.3.1 MapReduce-Based Design and Challenges

MapReduce is a programming model for processing large-scale data sets on clusters. With an associated implementation (e.g., Hadoop), MapReduce frees programmers from handling program's execution across a set of machines. It takes care of tasks scheduling, machine failures and inter-machine communication. A MapReduce algorithm has two phases: map that supports the distribution and partition of inputs to nodes, and `reduce` that groups and integrates the nodes' outputs. MapReduce data needs to be in the format of ⟨*key, value*⟩ pair, where *key* serves as an index and the value represents the properties corresponding to the key/data item. Programmer usually only needs to specify the `map` and `reduce` function to process the ⟨*key, value*⟩ pairs. Figure 5.2 illustrate the process of a MapReduce program execution.

The input big data in distributed file system is split and pre-processed by RECORDREADER. The output of RECORDREADER is a set of ⟨*key, value*⟩ pairs, which are sent to `map`. Each programmer specified map processes a ⟨*key, value*⟩ pair and generates a list of new ⟨*key, value*⟩ pairs. In Fig. 5.2, the first map generates three ⟨*key, value*⟩ pairs. The output ⟨*key, value*⟩ pairs are redistributed with the keys as indexes. All the pairs with key *K1* in Fig. 5.2 is processed by the first `reduce`. `Reduce` analyzes the group of values with the same key and writes the result back to distributed file system.

A significant of real world problems are able to be expressed by this model. A complex problem may require several rounds of map and reduce operations, requiring redefining and redistributing ⟨*key, value*⟩ pairs between rounds. New large-scale processing models are also proposed. For example, Google's Percolator [48] focuses on incrementally processing updates to a large data set. Muppet [38] provides a MapReduce-style model for streaming data. The collection intersection problem cannot be represented by the Percolator model. Although Muppet is able to perform collection intersection with streaming content, it cannot be used here due to its memory-heavy feature.

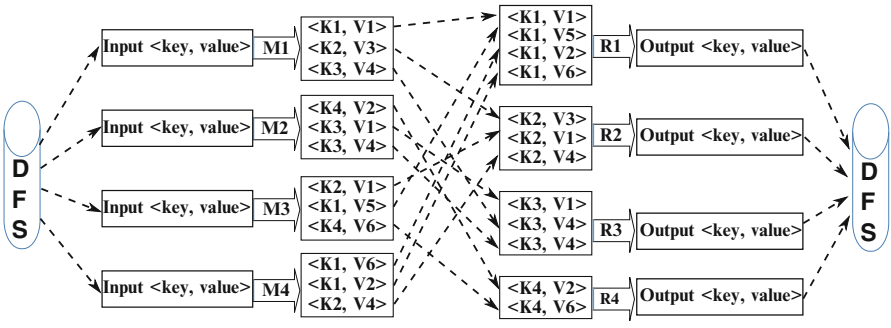

Fig. 5.2 Overview of MapReduce execution process. map takes each ⟨*key, value*⟩ pair as input and generates new ⟨*key, value*⟩ pairs. The output ⟨*key, value*⟩ pairs are redistributed according to the *key*. Each `reduce` processes the list of *value*s with the same key and writes results back to DFS

There exist several MapReduce-specific challenges when realizing collection-intersection based data-leak detection.

1. **Complex data fields** Collection intersection with duplicates is more complex than set intersection. This requires the design of complex data fields for ⟨*key, value*⟩ pairs and a series of map and `reduce` operations.
2. **Memory and I/O efficiency** The use of multiple data fields (e.g., collection size and ID, shingle frequency) in ⟨*key, value*⟩ pairs may cause frequent garbage collection and heavy network and disk I/O when processing big data.
3. **Optimal segmentation of data streams** While larger segment size allows the full utilization of CPU, it may cause insufficient memory problem and reduced detection sensitivity.

The MR-DLD data-leak detection algorithms in MapReduce addresses these technical challenges in MapReduce framework and achieves the security and privacy goals. The MR-DLD approach has well designed structured-yet-compact representations for data fields of intermediate values, which significantly improves the efficiency of the detection algorithms. The prototype also realizes an additional post-processing partitioning and analysis, which allows one to pinpoint the leak occurrences in large content segments. The MR-DLD approach is experimentally evaluated to test the impact of segment sizes on the detection throughput and identify the optimal segment size for performance.

5.3.3.2 Workload Distribution and Detection Workflow

The details of how the workload is distributed between data owner and DLD provider is as follows and shown in Fig. 5.3:

1. Data owner has m sensitive sequences $\{S_1, S_2, \cdots, S_m\}$ with average size $\mathscr{S}'$ and n content segments $\{C_1, C_2, \cdots, C_n\}$ with average size $\mathscr{C}'$. It obtains shingles from the content and sensitive data respectively. Then it chooses the parameters $(n, p(x), K, L)$, where n is the length of a shingle, $p(x)$ is the irreducible

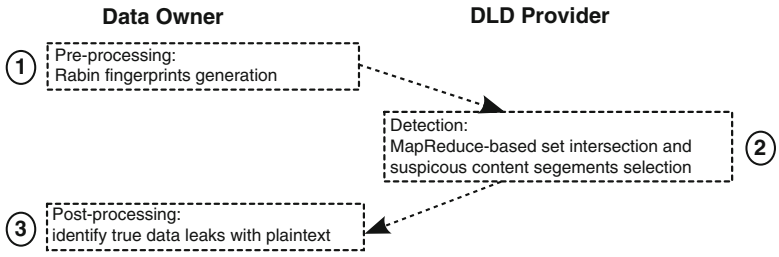

Fig. 5.3 Workload distribution for DLD provider and data owner

polynomial and L is the fingerprint length. The data owner computes Rabin fingerprints with (5.2) and releases the sensitive collections $\{C_{S1}, C_{S2}, \cdots, C_{Sm}\}$ and content fingerprint collections $\{C_{C1}, C_{C2}, \cdots, C_{Cn}\}$ to the DLD provider.

2. DLD provider receives both the sensitive fingerprint collections and content fingerprint collections. It deploys MapReduce framework and compares the n content collections with the m sensitive collections using two-phase MapReduce algorithms. By computing the intersection rate of each content and sensitive collections pair, it outputs whether the sensitive data was leaked and reports all the data-leak alerts to data owner.
3. Data owner receives the data-leak alerts with a set of tuples $\{(C_{Ci}, C_{Sj}), (C_{Ck}, C_{Sl}), \cdots\}$. The data owner maps them to suspicious content segments and the plain sensitive sequences tuples $\{(C_i, S_j), (C_k, S_l), \cdots\}$. The data owner consults plaintext content to confirm that true leaks (as opposed to accidental matches) occur in these content segments and further pinpoint the leak occurrences.

To compute the intersection rate of two fingerprint collections *Irate*, the MR-DLD approach has two MapReduce algorithms, DIVIDER and REASSEMBLER, each of which has a map and a `reduce` operation. Map and reduce operations are connected through a redistribution process. During the redistribution, outputs from map (in the form of $\langle key, value \rangle$ pairs) are sent to reducer nodes, as the inputs to the reduce algorithm. The key value of a record decides to which reducer node the it is forwarded. Records with the same key are sent to the same reducer.

Notations used in the algorithms introduced below are shown in Table 5.1, including collection identifier CID, size CSize (in terms of the number of items), occurrence frequency *Snum* of an item in one collection, occurrence frequency *Inum* of an item in an intersection, and intersection rate *Irate* of a content collection with respect to some sensitive data.

Table 5.1 Notations used in the MapReduce algorithms

Syntax	Definition
CID	An identifier of a collection (content or sensitive data)
CSize	Size of a collection
Snum	Occurrence frequency of an item
Inum	Occurrence frequency of an item in an intersection
CSid	A pair of CIDs $\langle CID_1, CID_2 \rangle$, where CID_1 is for a content collection and CID_2 is for a sensitive data collection
Irate	Intersection rate between a content collection and a sensitive data collection as defined in (5.1). Also referred to as the sensitivity score of the content.
ISN	A 3-item tuple of a collection ⟨identifier *CID*, size *CSize*, and the number of items in the collection⟩
CSS	An identifier for a collection intersection, consisting of an ID pair *CSid* of two collections and the size of the sensitive data collection *CSize*

1. DIVIDER takes the following as inputs: fingerprints of both content and sensitive data, and the information about the collections containing these fingerprints. Its purpose is to count the number of a fingerprint's occurrences in a collection intersection (i.e., *Inum* in (5.1)) for all fingerprints in all intersections.

   In map operation, it re-organizes the fingerprints to identify all the occurrences of a fingerprint across multiple content or sensitive data collections. Each map instance processes one collection. This reorganization traverses the list of fingerprints. Using the fingerprint as the key, it then emits (i.e., redistributes) the records with the same key to the same node.

   In `reduce`, for each fingerprint in an intersection the algorithm computes the *Inum* value, which is its number of occurrences in the intersection. Each `reduce` instance processes one fingerprint. The algorithm outputs the tuple $\langle CSS, Inum \rangle$, where *CSS* is the identifier of the intersection (consisting of IDs of the two collections and the size of the sensitive data collection.[1]) Outputs are written to MapReduce file system.
2. REASSEMBLER takes as inputs $\langle CSS, Inum \rangle$ (outputs from Algorithm DIVIDER). The purpose of this algorithm is to compute the intersection rates (i.e., *Irate* in (5.1)) of all collection intersections $\{C_{c_i} \cap C_{s_j}\}$ between a content collection C_{c_i} and a sensitive data collection C_{s_j}.

   In map, the inputs are read from the file system and redistributed to reducer nodes according to the identifier of an intersection *CSS* (key). A reducer has as inputs the *Inum* values for all the fingerprints appearing in a collection intersection whose identifier is *CSS*. At `reduce`, it computes the intersection rate of *CSS* based on (5.1).

In the next section, we present the algorithms for realizing the collection intersection workflow with one-way Rabin fingerprints. Section 5.3.5 explains why the privacy-preserving technique is able to protect the sensitive data against semi-honest MapReduce nodes.

5.3.4 Collection Intersection in MapReduce

We present the collection-intersection algorithm in the MapReduce framework to screening large content data. The algorithm computes the intersection rate of two collections as defined in (5.1). Each collection consists of Rabin fingerprints of *n*-grams generated from a sequence (sensitive data or content).

RECORDREADER is a (standard) MapReduce class. It is customized to read initial inputs into the detection system and transform them into the $\langle key, value \rangle$ format required by the map function. The initial inputs of RECORDREADER are content fingerprints segments and sensitive fingerprints sequences. For the DIVIDER

[1] The sensitive data collection is typically much smaller than the content collection.

Algorithm 1 DIVIDER: To count the number of a fingerprint's occurrences in a collection intersection for all fingerprints in all intersections

Input: Output of RECORDREADER in a format of ⟨*CSize, Fingerprint*⟩ as ⟨*key, value*⟩ pair.
Output: ⟨*CSS, Inum*⟩ as ⟨*key, value*⟩ pair, where *CSS* contains content collection ID, sensitive data collection ID and the size of the sensitive data collection. *Inum* is occurrence frequency of a fingerprint in the collection intersection.

```
1: function DIVIDER::MAPPER(CSize, Fingerprint)
2:     ▷ Record necessary information for the collection.
3:     ISN←CID, CSize and Snum
4:     Emit⟨Fingerprint, ISN⟩
5: end function
1: function DIVIDER::REDUCER(Fingerprint, ISNlist[c_1, ..., c_n])
2:     j = 0, k = 0
3:     ▷ Divide the list into a sensitive list and a content list
4:     for all c_i in ISNlist do
5:         if c_i belongs to sensitive collections then
6:             SensList[++j]← c_i
7:         else
8:             ContentList[++k] ← c_i
9:         end if
10:    end for
11:    ▷ Record the fingerprint occurrence in the intersection
12:    for all sens in SensList do
13:        for all content in ContentList do
14:            Size ←sens.CSize
15:            Inum ←Min(sens.Snum, content.Snum)
16:            CSS← ⟨content.CID, sens.CID, Size⟩
17:            Emit ⟨CSS, Inum⟩
18:        end for
19:    end for
20: end function
```

algorithm, the RECORDREADER has two tasks: (1) to read in each map split (e.g., content segment) as a whole and (2) to generate ⟨*CSize, fingerprint*⟩ pairs required by the map operation of DIVIDER algorithm.

5.3.4.1 DIVIDER Algorithm

DIVIDER is the most important and computational intensive algorithm in the system. Pseudocode of DIVIDER is given in Algorithm 1. In order to count the number of a fingerprint's occurrences in a collection intersection, the map operation in DIVIDER goes through the input ⟨*CSize, fingerprint*⟩ pairs, and reorganizes them to be indexed by fingerprint values. For each fingerprint in a collection, map records its origin information (e.g., *CID, CSize* of the collection) and *Snum* (fingerprint's frequency of occurrence in the collection). These values are useful for later intersection-rate

computation. The advantage of using the fingerprint as the index (key) in the map's outputs is that it allows the reducer to quickly identify non-intersected items.

After redistribution, entries having the same fingerprint are sent to the same reducer node as inputs to the reduce algorithm.

Reduce algorithm is more complex than map. It partitions the occurrences of a fingerprint into two lists, one list (*ContentList*) for the occurrences in content collections and the other for sensitive data (*SensList*). It then uses a double for-loop to identify the fingerprints that appear in intersections. Non-intersected fingerprints are not analyzed, significantly reducing the computational overhead. This reduction is reflected in the computational complexity analysis in Table 5.2, specifically the $\gamma \in [0, 1]$ reduction factor representing the size of the intersection.

The for-loops also compute the occurrence frequency *Inum* of the fingerprint in an intersection. The output of the algorithm is the $\langle CSS, Inum \rangle$ pairs, indicating that a fingerprint occurs *Inum* number of times in a collection intersection whose identifier is *CSS*.

5.3.4.2 REASSEMBLER Algorithm

The purpose of REASSEMBLER is to compute the intersection rates *Irate* of all collection-and-sensitive-data intersections. Pseudocode of REASSEMBLER is in Algorithm 2. The map operation in REASSEMBLER emits (i.e., redistributes) inputs

Algorithm 2 REASSEMBLER: To compute the intersection rates *Irate* of all collection intersections $\{C_{c_i} \cap C_{s_j}\}$ between a content collection C_{c_i} and a sensitive data collection C_{s_j}

Input: Output of DIVIDER in a format of $\langle CSS, Inum \rangle$ as $\langle key, value \rangle$ pairs.
Output: $\langle CSid, Irate \rangle$ pairs where *CSid* represents a pair of a content collection ID and a sensitive collection ID, while *Irate* represents the intersection rate between them

```
1: function REASSEMBLER::MAPPER(CSS, Inum)
2:     Emit⟨CSS, Inum⟩
3: end function
```

```
1: function REASSEMBLER::REDUCER(CSS, Inum[n_1, ..., n_n])
2:     intersection ← 0
3:     ▷ Add up all the elements in Inum[]
4:     for all n_i in Inum[] do
5:         intersection ← intersection + n_i
6:     end for
7:     CSid ← CSS.CSid
8:     ▷ Compute intersection rate
9:     Irate ← |intersection| / CSS.CSize
10:    Emit ⟨CSid, Irate⟩
11: end function
```

⟨*CSS, Inum*⟩ pairs according to their key *CSS* values to different reducers. The reducer can then compute the intersection rate *Irate* for the content and sensitive data collection pair. I.e., this redistribution sends all the intersected items between a content collection C_{c_i} and a sensitive data collection C_{s_j} to the same reducer.

5.3.4.3 Example of the Algorithms

Steps of the MapReduce algorithms are illustrated with an example (on four MapReduce nodes) in Fig. 5.4. The example has two content collections C_1 and C_2, and two sensitive data collections S_1 and S_2. The four data collections are generated by the data owner and sent to DLD provider. The sizes of the corresponding collections are 3, 4, 3 and 3, respectively. Each element (e.g., a) in the collections represents a fingerprint. The items after the steps indicate how the operations compute.

Step 1 Before the algorithms, the customized RECORDREADER reads the collections and sends ⟨*key, value*⟩ pairs to maps. In node 1, RECORDREADER parses collection C_1 by generating a ⟨*key, value*⟩ whenever it encounters an element. The key is the collection size 3 for C_1 and the value is the element it encounters.

- Node1: $\{a, b, c\} \Rightarrow \{\langle 3, a\rangle, \langle 3, b\rangle, \langle 3, c\rangle\}$
- Node2: $\{a, h, c, h\} \Rightarrow \{\langle 4, a\rangle, \langle 4, h\rangle, \langle 4, c\rangle, \langle 4, h\rangle\}$
- Node3: $\{a, b, d\} \Rightarrow \{\langle 3, a\rangle, \langle 3, b\rangle, \langle 3, d\rangle\}$
- Node4: $\{d, h, h\} \Rightarrow \{\langle 3, d\rangle, \langle 3, h\rangle, \langle 3, h\rangle\}$

Step 2 For the element a in node 1, map in DIVIDER outputs the pair $\langle a, (C_1, 3, 1)\rangle$, indicating that fingerprint (key) a is from content collection C_1 of size 3 and occurs once in C_1. The outputs are redistributed according the key values. All occurrences of fingerprint a are sent to node 1, including two occurrences

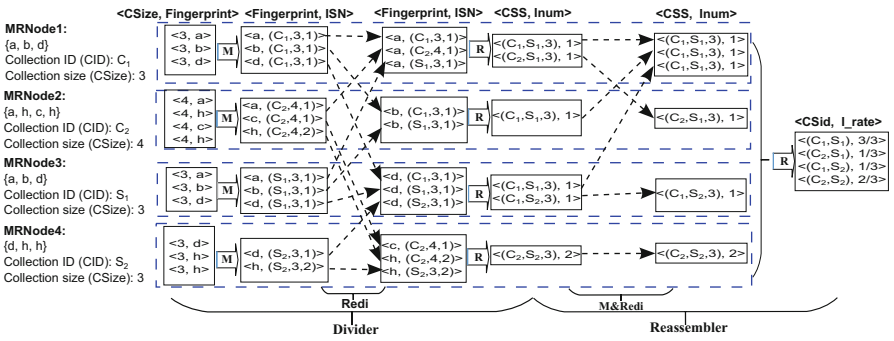

Fig. 5.4 An example illustrating DIVIDER and REASSEMBLER algorithms, with four MapReduce nodes, two content collections C_1 and C_2, and two sensitive data collections S_1 and S_2. **M**, **R**, **Redi** stand for map, reduce, and redistribution, respectively. ⟨key, value⟩ of each operation is shown at the *top*

from content collections C_1 and C_2, one occurrence from sensitive data collection S_1. Similar process applies to all the other fingerprints. The items below shows how a is manipulated in different nodes.

- a in node1: $\langle 3, a\rangle \Rightarrow \langle a, (C_1, 3, 1)\rangle$
- a in node2: $\langle 4, a\rangle \Rightarrow \langle b, (C_2, 4, 1)\rangle$
- a in node3: $\langle 3, a\rangle \Rightarrow \langle c, (S_1, 3, 1)\rangle$

Step 3 `Reduce` in DIVIDER computes the intersection of content and sensitive collections for each fingerprint. In node 1, given the list of collections that fingerprint a exists, reduce algorithm uses a double for-loop and identifies that a appears in intersection $C_1 \cap S_1$ and intersection $C_2 \cap S_1$. The intersections are set as keys. The occurrence frequencies of fingerprint a are set as values. In node 1, a appears once in $C_1 \cap S_1$ and once in $C_2 \cap S_1$.

- Node1: $\langle a, \{(C_1, 3, 1), (C_2, 4, 1), (S_1, 3, 1)\}\rangle \Rightarrow \{\langle (C_1, S_1, 3), 1\rangle, \langle (C_2, S_1, 3), 1\rangle\}$
- Node2: $\langle b, \{(C_1, 3, 1), (S_1, 3, 1)\}\rangle \Rightarrow \{\langle (C_1, S_1, 3), 1\rangle\}$
- Node3: $\langle d, \{(C_1, 3, 1), (S_1, 4, 1), (S_2, 3, 1)\}\} \Rightarrow \{\langle (C_1, S_1, 3), 1\rangle, \langle (C_1, S_2, 3), 1\rangle\}$
- Node4: $\langle c, \{(C_2, 4, 1)\}\rangle \Rightarrow NULL$; $\langle h, \{(C_2, 4, 2), (S_2, 3, 2)\}\rangle \Rightarrow \{\langle (C_2, S_2, 3), 2\rangle\}$

Step 4 In REASSEMBLER, the outputs of DIVIDER are redistributed. All the pairs with the same intersection are sent to the same node. In node 1, all the occurrence frequencies of fingerprints in intersection $C_1 \cap S_1$ are collected. The total number of fingerprints shared by C_1 and S_1 is 3. The intersection rate is 1.

- Node1: $\langle (C_1, S_1, 3), \{1, 1, 1\}\rangle \Rightarrow \langle (C_1, S_1), 3/3\rangle$
- Node2: $\langle (C_2, S_1, 3), \{1\}\rangle \Rightarrow \langle (C_2, S_1), 1/3\rangle$
- Node3: $\langle (C_1, S_2, 3), \{1\}\rangle \Rightarrow \langle (C_1, S_2), 1/3\rangle$
- Node4: $\langle (C_2, S_2, 3), \{2\}\rangle \Rightarrow \langle (C_2, S_2), 2/3\rangle$

With the outputs of all the nodes, we can get the intersection rates of all the intersections. The intersection rates are used to determine which content collections are suspicious.

5.3.4.4 Complexity Analysis

The computational and communication complexities of various operations of the algorithm are shown in Table 5.2. The average size of a sensitive data collection is denoted by $\mathscr{S}$, the average size of a content collection by $\mathscr{C}$, the number of sensitive data collections by m, the number of content collections by n, and the average intersection rate by $\gamma \in [0, 1]$. In real world detection, the size of $\mathscr{C}m$ could be very large. Without loss of generality, it is assumed that $|\mathscr{S}| < |\mathscr{C}|$ and $|\mathscr{S}m| < |\mathscr{C}n|$. Post-processing is not included in complexity analysis.

Table 5.2 Computation and communication complexity of each phase in the MR-DLD MapReduce algorithm and that of the conventional hashtable-based approach

Algorithm	Computation	Communication
Pre-processing of MR-DLD	$O(\mathscr{C}n + \mathscr{S}m)$	$O(\mathscr{C}n + \mathscr{S}m)$
Divider of MR-DLD::Mapper	$O(\mathscr{C}n + \mathscr{S}m)$	$O(\mathscr{C}n + \mathscr{S}m)$
Divider of MR-DLD::Reducer	$O(\mathscr{C}n + \mathscr{S}mn\gamma)$	$O(\mathscr{S}mn\gamma)$
Reassembler of MR-DLD::Mapper	$O(\mathscr{S}mn\gamma)$	$O(\mathscr{S}mn\gamma)$
Reassembler of MR-DLD::Reducer	$O(\mathscr{S}mn\gamma)$	$O(mn)$
Total of MR-DLD	$O(\mathscr{C}n + \mathscr{S}mn\gamma)$	$O(\mathscr{C}n + \mathscr{S}mn\gamma)$
Hashtable	$O(\mathscr{C}n + \mathscr{S}mn)$	N/A

The average size of a sensitive data collection is denoted by $\mathscr{S}$, the average size of a content collection by $\mathscr{C}$, the number of sensitive data collections by m, the number of content collections by n, and the average intersection rate by $\gamma \in [0, 1]$

The total communication complexity $O(\mathscr{C}n + \mathscr{S}mn\gamma)$ covers the number of records ($\langle$*key, value*$\rangle$ pairs) that all operations output. For a hashtable-based (non-MapReduce) approach, where each content collection is stored in a hashtable (total n hashtables of size $\mathscr{C}$ each) and each sensitive data item (total $\mathscr{S}m$ items) is compared against all n hashtables, the computational complexity is $O(\mathscr{C}n + \mathscr{S}mn)$.

5.3.5 Security Analysis and Discussion

MapReduce nodes that perform the data-leak detection may be controlled by honest-but-curious providers (aka semi-honest), who follow the protocol, but may attempt to gain knowledge of the sensitive data information (e.g., by logging the intermediate results and making inferences). The security and privacy guarantees provided by the MR-DLD data-leak detection system is analyzed in this subsection. We also point out the limitations associated with the collection intersection based DLD approach.

5.3.5.1 Privacy Guarantee

The privacy goal of the MR-DLD system is to prevent the sensitive data from being exposed to DLD provider or untrusted nodes. Let f_s be the Rabin fingerprint of sensitive data shingle s. Using the algorithms in Sect. 5.3.4, a MapReduce node knows fingerprint f_s but not shingle s of the sensitive data. Attackers are assumed to be not able to infer s in polynomial time from f_s. This assumption is guaranteed by the one-way Rabin fingerprinting function [51].

In addition, to prevent DLD provider from performing frequency analysis, the data owner chooses a different irreducible polynomial $p(x)$ for each session. To be specific, the data owner needs to:

1. Divide the content into multiple blocks. Each block is assigned to a session.
2. Select a new irreducible polynomial $p(x)$ for each session.
3. Divide each session (block) into several subsessions. Select a new secret session key K for each subsession. We assume that the data in each subsession is small enough, without providing useful frequency information.
4. For each subsession, pre-process the content block and a copy of sensitive data with the polynomial $p(x)$. XOR all the fingerprints of the subsession with session key K and send the result to DLD provider.

This above transformation preserves the equality comparison (as required in Sect. 5.3.2). It ensures that the same shingles are mapped to the same fingerprint within a session).

Following the above steps, the same shingle is mapped to different fingerprints in multiple sessions. The advantage of this design is the increased randomization in the fingerprint computation, making it more challenging for the DLD provider to correlate values and infer preimage. DLD provider cannot have enough frequency information to infer sensitive information. This randomization also increases the difficulty of dictionary attacks.

In MR-DLD system, the irreducible polynomials need to be only known to the data owner. As Rabin fingerprint is not designed to be a keyed hash, DLD provider may still be able to infer the polynomial from fingerprints. However, the privacy of sensitive data is still guaranteed even if $p(x)$ is known to DLD provider. Let f_1 be the Rabin fingerprint of shingle c_1 in block B_1 and f_2 be the Rabin fingerprint of shingle c_2 in block B_2. B_1 uses irreducible polynomial $p(x)_1$ and $B2$ uses irreducible polynomial $p(x)_2$. DLD provider can merge the frequency information of block B_1 and block B_2 only if it knows whether f_1 and f_2 are generated from the same shingle. However, this is computational impossible because DLD provider does not know the session keys for the two blocks, thus needs to resort to brute-force guessing, which is expensive.

To perform successful frequency analysis, the DLD provider needs to have a large content block, which is transformed with one key K and one $p(x)$. The transformation breaks large content into smaller ones, with each encoded with different key K and $p(x)$. As the DLD provider cannot merge frequency information of multiple blocks, the combination usage of XOR and $p(x)$ can help increases the difficulty of successful frequency analysis.

5.3.5.2 Detection Accuracy

We discuss the sources of possible false negatives—data-leak cases being overlooked and false positives—legitimate content misclassified as data leak in the detection.

Collisions Collisions may be due to where the legitimate content happens to contain the partial sensitive-data fingerprints by coincidence. The collisions may increase with shorter shingles, or smaller numbers of partial fingerprints, and may

decrease if additional features such as the order of fingerprints are used for detection. A previous large-scale information-retrieval study empirically demonstrated the low rate of this type of collisions in Rabin fingerprint [8], which is a desirable property suggesting low unwanted false alarms in our DLD setting. Using 6 shingles of 8 bytes each on 200 million documents, researchers found that the probability for two documents that share more than a certain number of shingles to significantly differ from each other is quite low [8]. For example, the probability that two documents having resemblance greater than 97.5 % do not share at least two features is less than 0.01; and the probability that two documents with less than 77 % resemblance do share two or more shingles is less than 0.01 [8]. Collisions due to two distinct shingles generating the same fingerprint are proved to be low [7] and are negligible.

Modified data leak The underlying shingle scheme of the basic approach has limited power to capture heavily modified data leaks. False negatives (i.e., failure to detect data leak) may occur due to the data modification (e.g., reformatting). The new shingles/fingerprints may not resemble the original ones, and cannot be detected. As a result, a packet may evade the detection. The modified data-leak detection problem is a general problem for all comparison-based data-leak detection solutions. More advanced content comparison techniques than shingles/fingerprints are needed to fully address the issue.

Selective fragments leak The partial disclosure scheme may result in false negatives, i.e., the leaked data may evade the detection because it is not covered by the released fingerprints. This issue illustrates the tradeoff among detection accuracy, privacy guarantee and detection efficiency.

5.3.6 Evaluation

The algorithms are implemented with Java in Hadoop, which is an open-source software system implementing MapReduce. The length of fingerprint and shingle is set to 8 bytes (64 bits). This length was previously reported as optimal for robust similarity test [8], as it is long enough to preserve some local context and short enough to resist certain transformations. The prototype also implements an additional IP-based post-processing analysis and partition focusing on the suspicious content. It allows the data owner to pinpoint the IPs of hosts where leaks occur. The outputs are the suspicious content segments and corresponding hosts.

Several technical measures are made to reduce disk and network I/O. SequenceFile (structured) format is used as the intermediate data format. The size of ⟨*key, value*⟩ pairs is also minimized. E.g., the size of *value* after map in DIVIDER is 6 bytes on average. COMBINATION classes are also implemented to significantly reduce the amount of intermediate results written to the distributed file systems (DFS). This reduction in size is achieved by aggregating same ⟨*key, value*⟩ pairs. This method reduces the data volume by half. Hadoop compression is also enabled, giving as high as 20-fold size reduction.

The algorithms are deployed in two different 24-node Hadoop systems, a local cluster and Amazon Elastic Compute Cloud (EC2). For both environments, one node is set as the master node and the rest as slave nodes.

- *Amazon EC2:* 24 nodes each having a c3.2xlarge instance with eight CPUs and 15 GB RAM.
- *Local cluster:* 24 nodes each having two quad-core 2.8 GHz Xeon processors and 8 GB RAM.

Enron Email Corpus, including both email header and body, are used to perform the performance experiments. The entire dataset is used as content and a small subset of it is used as the sensitive data.

The experiments aim to answer the following questions.

1. How does the size of content segment affect the analysis throughput? (Sect. 5.3.6.1)
2. What is the throughput of the detection on Amazon EC2 and the local clusters? (Sect. 5.3.6.2)
3. How does the size of sensitive data affect the detection performance? (Sect. 5.3.6.3)

5.3.6.1 Optimal Size of Content Segment

Content volume is usually overwhelmingly larger than sensitive data, as new content is generated continuously in storage and in transmission. Thus, the throughput of different sizes and numbers of content segments is evaluated in order to find the optimal segment size for scalability on DLD provider's side. A content segment with size $\hat{C}$ is the original sequence that is used to generate the n-gram content collection. A sensitive sequence with size $\hat{S}$ is the original sequence that is used to generate the n-gram sensitive collection.

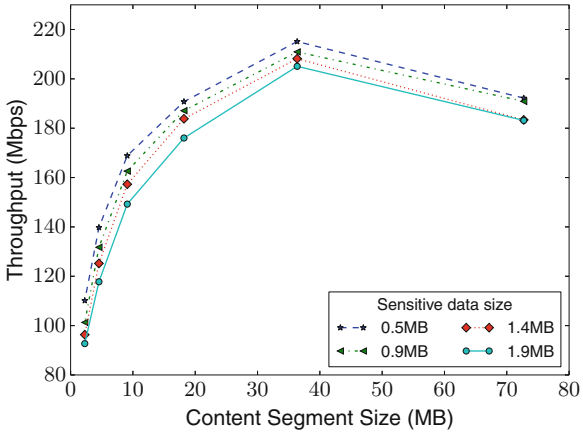

Fig. 5.5 DLD provider's throughput with different sizes of content segments. For each setup (*line*), the size of the content analyzed is 37 GB

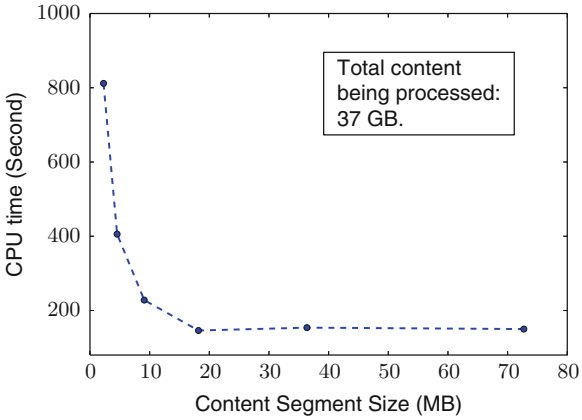

Fig. 5.6 Data owner's pre-processing overhead on a workstation with different sizes of content segments. The total size of content analyzed is 37 GB

The total size of content analyzed is 37 GB, which consists of multiple copies of Enron data. Detection performance under different content segment sizes (from 2 to 80 MB) is measured. The size of sensitive data is varied from 0.5 to 2 MB. The results are shown in Fig. 5.5.

We can observe that when $\hat{C} < 37$ MB, the throughput of the analysis increases with the size $\hat{C}$ of content segment. When $\hat{C}$ becomes larger than 37 MB, the throughput begins to decrease. The reason for this decrease is that more computation resources are spent on garbage collection with larger $\hat{C}$. There are over 16 processes running at one node at the same time. Each process is assigned 400 MB to process 37×8 MB shingles.

Data owner's overhead of generating fingerprints is also evaluated with the same datasets. The experiment is performed on a quad-core 3.00 GHz Xeon processor and 8 GB RAM machine running Ubuntu 14.04.1. The results are shown in Fig. 5.6. We observe that the CPU time of generating fingerprints falls off quickly with the content segment size increasing. When $\hat{C} > 20$ MB, the CPU time is less than 100 s, meaning that the throughput is more than 3000 Mbps. The total time of pre-processing is over 40 min due to the speed limit of I/O. However, pre-processing can be easily parallelized. The time of generating fingerprints is linearly decreased with multiple machines processing the content at the same time. This fact indicates that the data owner can handle the fingerprint generation process without significant overhead incurred.

Thus, the size of content segments is set to 37 MB for the rest of the experiments. This size also allows the full utilization of the Hadoop file system (HDFS) I/O capacity without causing out-of-memory problems.

5.3.6.2 Scalability

For scalability evaluation, 37 GB content is processed with different numbers of nodes, 4, 8, 12, 16, 20, and 24. The experiments were deployed both on the local

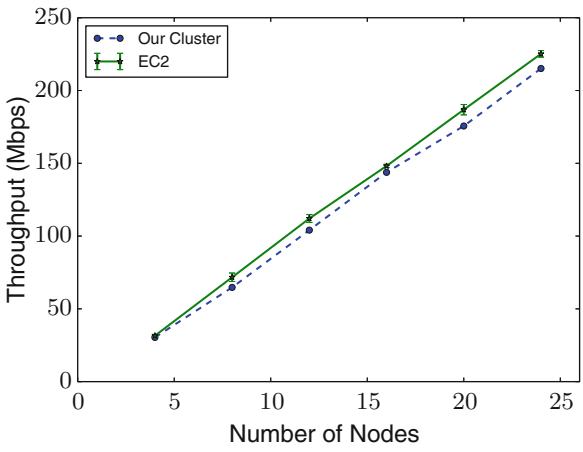

Fig. 5.7 Throughput with different number of nodes on a local cluster or Amazon EC2

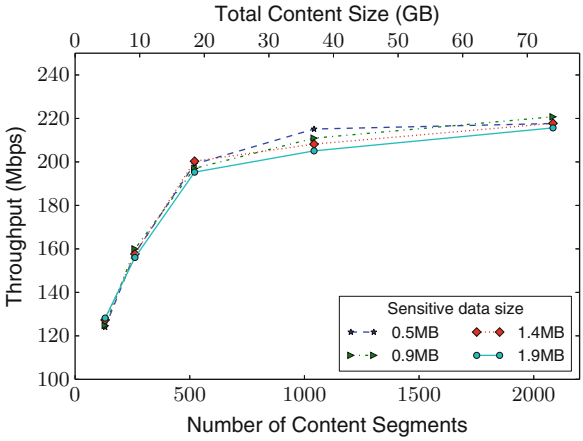

Fig. 5.8 Throughput with different amount of content workload. Each content segment is 37 MB

cluster and on Amazon EC2. The results are shown in Fig. 5.7. The system scales well, as the throughput linearly increases with the number of nodes. The peak throughput observed is 215 Mbps on the local cluster and 225 Mbps on Amazon EC2. EC2 cluster gives 3–11 % performance improvement. This improvement is partly due to the larger memory. The standard error bars of EC2 nodes are shown in Fig. 5.7 (from three runs). Variances of throughputs on the local cluster are negligible.

The throughput is also evaluated under a varying number of content segments n, i.e., workload. The results are shown in Fig. 5.8, where the total size of content analyzed is shown at the top X-axis (up to 74 GB). Throughput increases as workload increases as expected.

In the experiments above, the size of sensitive data is small enough to fit in one collection. The larger size of sensitive data increases the computation overhead, which explains the slight decrease in throughput in Figs. 5.5 and 5.8.

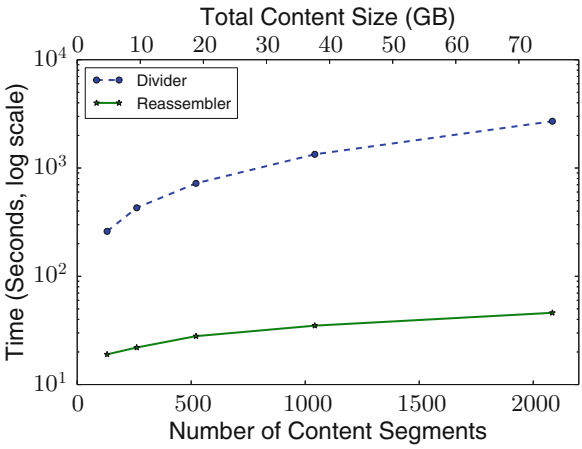

Fig. 5.9 Runtime of DIVIDER and REASSEMBLER algorithms. The DIVIDER operation takes 85–98 % of the total runtime. The Y-axis is in 10 based log scale

The total overhead are broken down based on the DIVIDER and REASSEMBLER operations. The results are shown in Fig. 5.9 with the runtime (Y-axis) in a log scale. DIVIDER algorithm is much more expensive than REASSEMBLER, accounting for 85–98 % of the total runtime. With increasing content workload, DIVIDER's runtime increases, more significantly than that of REASSEMBLER.

These observations are expected, as DIVIDER algorithm is more complex. Specifically, both map and `reduce` in DIVIDER need to touch *all* content items. Because of the large content volume, these operations are expensive. In comparison, REASSEMBLER algorithm only touches the intersected items, which is substantially smaller for normal content without leaks. These experimental results are consistent with the complexity analysis in Table 5.2.

5.3.6.3 Performance Impact of Sensitive Data

The performance results in Fig. 5.5 are reorganized so that the sizes of sensitive data are shown at the X-axis. The new figure is Fig. 5.10, where each setup (line) processes 37 GB data and differs in their size for content segment. There are a few observations. First, smaller content segment size incurs higher computational overhead, e.g., for keeping tracking the collection information (discussed in Sect. 5.3.6.1).

The second observation is that the runtime increases as the size of sensitive data increases, which is expected. Experiments with the largest content segment (bottom line in Fig. 5.10) have the smallest increase, i.e., the least affected by the increasing volume of sensitive data.

This difference in intercept is explained next. The total computation complexity is $O(\mathscr{C}n + \mathscr{S}mn\gamma)$ (Table 5.2). In $O(\mathscr{C}n + \mathscr{S}mn\gamma)$, $n\gamma$ serves as the coefficient (i.e., intercept), as the total size $\mathscr{S}m$ of sensitive data increases, where n is the number of content segments. When 37 GB content is broken into small segments,

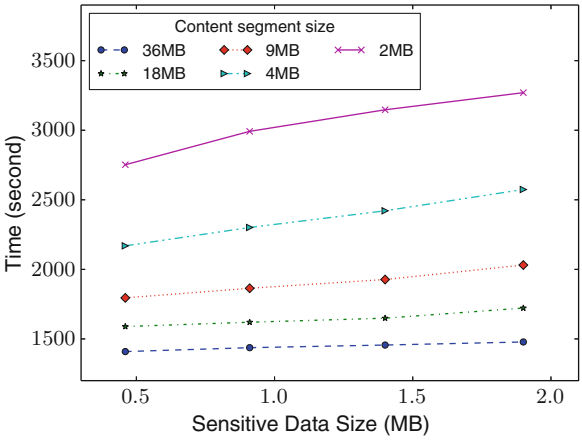

Fig. 5.10 Runtime with a varying size of sensitive data. The content volume is 37.5 GB for each setup. Each setup (*line*) has a different size for content segments

n is large. A larger coefficient magnifies the increase in sensitive data, resulting in more substantial overhead increase. Therefore, the line at the bottom of Fig. 5.10 represents the recommended configuration with a large 37 MB content segment size.

Summary The experimental findings are summarized as below.

1. The MR-DLD approach linearly scale on big data with the number of nodes. it achieves 225 Mbps throughput on Amazon EC2 cluster and a similar throughput on the local cluster. *Divider* algorithm accounts for 85–98 % of the total runtime.
2. Larger content segment size $\hat{C}$ (up to 37 MB) gives higher performance. This observation is due to the decreased amount of bookkeeping information for keeping track of collections, which results in significantly reduced I/O overhead associated with intermediate results. Data owner can also handle the fingerprint generation process without significant overhead incurred.
3. When the content segment size $\hat{C}$ is large (37 MB), the increase in the amount of sensitive data has a relatively small impact on the runtime. Given the content workload, larger $\hat{C}$ means fewer number of content segments, resulting in a smaller coefficient.

5.4 AlignDLD: Data-Leak Detection Through Alignment

To address the detection accuracy challenges, i.e., transformed data leaks and partial data leaks, we present *AlignDLD*, a data-leak detection solution with alignment techniques [58]. The key to the detection of transformed data leaks is a specialized sequence alignment algorithm, which handles arbitrary variations of patterns without an explicit specification of all possible variation patterns.

AlignDLD conducts an alignment 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.

Traditional alignment algorithms based on dynamic programming is slow and cannot be directly applied to data-leak detection tasks on big data. In order to address the scalability issue, AlignDLD consists of a *comparable* sampling algorithm and a *sampling oblivious* alignment algorithm. The 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. AlignDLD aligns sampled sequences to infer the similarity between the original sequences before sampling.

Experiments show that AlignDLD achieves accurate detection with low false positive and false negative rates. It substantially outperforms traditional collection-intersection methods in terms of detection accuracy. It also meets the scalability requirement for data-leak detection tasks on big data.

5.4.1 Models and Overview

Traditional collection-based data-leak detection models face two challenges toward accurate data-leak detection.

High detection specificity: the ability to distinguish true leaks from coincidental matches, which can cause false alarms. Existing collection-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.

We present AlignDLD, a data-leak detection model using efficient sequence comparison techniques to analyze a large amount of content. A diagram illustrating the security model of AlignDLD is shown in Fig. 5.11.

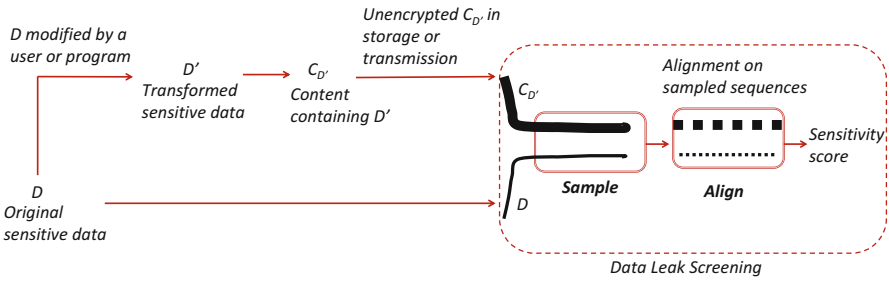

Fig. 5.11 A schematic drawing showing the two types of sequences in transformed data-leak detection model, their relations, and the workflow of the detection

AlignDLD 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.

The workflow includes EXTRACTION, PREPROCESSING, SAMPLING, ALIGNMENT, and DECISION operations.

EXTRACTION collects the content sequences.
PREPROCESSING prepares the sequences of n-grams for content/sensitive data.
SAMPLING generates samples from the preprocessed content/sensitive sequences.
ALIGNMENT performs local alignment between the two sampled sequences.
DECISION confirms and reports leaks according to the sensitive data sharing policy.

5.4.2 *Comparable Sampling*

We formally define the new sampling requirement needed in data-leak detection and present the comparable sampling algorithm used in AlignDLD.

5.4.2.1 Definitions

One great challenge in aligning sampled sequences is that the sensitive data segment can be exposed at an arbitrary position in a network traffic stream or a file system. The sampled sequence should be deterministic despite the starting and ending points of the sequence to be sampled. Moreover, the leaked sensitive data could be inexact but similar to the original string due to unpredictable transformations. We define the capability of giving comparable results from similar strings in Definition 1.

Definition 1. (Comparable sampling) if string x is similar to a substring of string y according to a similarity measure M, then x' (the sampled subsequences[2] of x) is similar to a substring of y' (the sampled subsequence of y) according to M.

If we restrict the similarity measure M in Definition 1 to *identical relation*, we get a specific instance of comparable sampling in Definition 2.

Definition 2. (Subsequence-preserving sampling) if string x is a substring of string y, then x' is also a substring of y', where x' is a sampled subsequence of x, and y' is a sampled subsequence of y.

Because a subsequence-preserving sampling procedure is a restricted comparable sampling, so the subsequence-preserving sampling is deterministic, i.e., the same input always yields the same output. The vice versa may not be true.

5.4.2.2 Comparable Sampling Algorithm

We present a comparable sampling algorithm, the advantage of which 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, the sampling algorithm is *deterministic* and *subsequence-preserving*.

The comparable sampling algorithm takes in $\mathscr{S}$, an input list of items (n-grams of sensitive data or content), and outputs $\mathscr{T}$, 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 $\mathscr{T}$ can be aggregated, and $\mathscr{T}$ can be turned into a *compact representation* $\mathscr{L}$. Each item in $\mathscr{L}$ contains the value of the sampled item and the length of the null region between the current sampled item and the preceding one.

$\mathscr{T}$ is initialized as an empty list, i.e., a list of null items. The algorithm runs a small sliding window w on $\mathscr{S}$ and utilizes a selection function to decide what items in w should be selected for $\mathscr{T}$. The selection decision 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, Algorithm 3 satisfies the subsequence-preserving and comparable sampling requirements.

In Algorithm 3, without loss of generality, we describe the 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. f is *deterministic*. It selects items without bias when items (n-grams) are preprocessed using Rabin's fingerprint.[3] f can be replaced by other functions that have the same properties. The selection results at each sliding window position determine what items are chosen for the sampled list. The parameters N and $|w|$ determine the sampling rate. The directional collection difference operation

[2]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).

[3]Rabin's fingerprint is min-wise independent.

Algorithm 3 A subsequence-preserving sampling algorithm

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

`collectionDiff` in Algorithm 3 (lines 10 and 11) is similar to the set difference operation, except that it does not eliminate duplicates, i.e., identical items are kept in the results.

$\mathscr{T}$ output by Algorithm 3 takes the same space as $\mathscr{S}$ does. Null items can be combined, and $\mathscr{T}$ is turned into a *compact representation* $\mathscr{L}$, which is consumed by the sampling-oblivious alignment algorithm in the next phase.

5.4.2.3 Sampling Algorithm Analysis

The complexity of Algorithm 3 using the $\min(w, N)$ selection function is $O(n \log |w|)$, or $O(n)$ where n is the size of the input, $|w|$ is the size of the window. The factor $O(\log |w|)$ comes from maintaining the smallest N items within the window w.

The sampling rate $\alpha \in [\frac{N}{|w|}, 1]$ approximates $\frac{N}{|w|}$ for random inputs. For arbitrary inputs, the actual sampling rate depends on the characteristics of the input space and the selection function used.

A sufficient number of items need to be sampled from sequences to warrant an accurate detection. The empirical result in Sect. 5.4.4 shows that sampling with $\alpha = 0.25$ can detect as short as 32-byte-long sensitive data segments.

5.4.3 Sampling Oblivious Alignment

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

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. The described algorithm infers the comparison outcomes between null regions based on (1) the comparison outcomes between items surrounding null regions and (2) sizes of null regions, because leaked data region is usually consecutive, e.g., spans at least dozens of bytes. 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.

5.4.3.1 Dynamic Programming Components

The presented sample-oblivious alignment algorithm is based on 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 the 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 the dynamic program alignment algorithm in Algorithm 4. For the i-th item $\mathscr{L}_i$ in a sampled sequence $\mathscr{L}$ (the compact form), the field $\mathscr{L}_i.value$ denotes the value of the item and a new field $\mathscr{L}_i.span$ denotes the size of null region between that item and the preceding non-null item. The algorithm computes a non-negative score matrix H of size $|\mathscr{L}^a|$-by-$|\mathscr{L}^b|$ for the input sequence $\mathscr{L}^a$ and $\mathscr{L}^b$ 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.

The weight function $f_w()$ takes three inputs: the two items being aligned (e.g., $\mathscr{L}_i^a$ from sensitive data sequence and $\mathscr{L}_j^b$ 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 $\mathscr{L}_i^a$ and $\mathscr{L}_j^b$ 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 4 differs

Algorithm 4 Recurrence relation in dynamic programming

Require: 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 $\mathscr{L}_i^a, \mathscr{L}_j^b$ in two sampled sequences $\mathscr{L}^a$ and $\mathscr{L}^b$, respectively.

Ensure: $H(i,j)$

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

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

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

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

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

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

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

from the typical weight function in Smith-Waterman algorithm [59] 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.

5.4.3.2 Alignment Algorithm Analysis

The complexity of the presented alignment algorithm is $O(|\mathscr{L}^a||\mathscr{L}^b|)$, where $|\mathscr{L}^a|$ and $|\mathscr{L}^b|$ are lengths of compact representations of the two sampled sequences. The alignment complexity for a single piece of sensitive data of size l is the same as that of a set of shorter pieces with a total size l, as the total amounts of matrix cells to compute are the same.

In a real-world deployment, the overall sensitive data sequence $\mathscr{S}^a$ is usually close to a fixed length, and more attention is commonly paid to the length of the content sequence $\mathscr{S}^b$. In this case, the complexity of the alignment is $O(|\mathscr{L}^b|)$ where $\mathscr{L}^b$ is the sampled list of $\mathscr{S}^b$.

The alignment of two sampled sequences achieves a speedup in the order of $O(\alpha^2)$, where $\alpha \in (0,1)$ is the sampling rate. There is a constant damping factor due to the overhead introduced by sampling. The expected value is 0.33 because of the extra two fields, besides the score field, to maintain for each cell in H. The damping factor is experimentally verified in Sect. 5.4.4.

5.4.4 Detection Accuracy Evaluation

We present the detection results of AlignDLD against transformed data leaks and partial data leaks in this section. AlignDLD is compared with Coll-Inter, a standard

collection-intersection method. Parallel versions of the prototype are discussed in Sect. 5.4.5 to demonstrate the detection capability in processing big data.

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

5.4.4.1 Detecting Modified Leaks

Data-leak detection are performed via 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., genuine sensitive data appears in the content.
3. Content with modified leaks caused by `WordPress`, which substitutes every space with "+" in the content.

The content and sensitive data in this experiment are selected emails from *Enron* dataset, which consists of 517,424 real emails from 150 users [34]. 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 (5.4).

$$\mathbb{S} = \frac{\xi}{r \times \min\left(|\mathscr{S}^a|, |\mathscr{S}^b|\right)} \tag{5.4}$$

The results of the detection are presented in Fig. 5.12. 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 Fig. 5.12 represent the detection results of leaks with `WordPress` modifications. AlignDLD in Fig. 5.12a gives much higher sensitivity scores to the transformed data leak than Coll-Inter. 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, Coll-Inter in Fig. 5.12b yields a significant overlap between messages with no leak and messages with transformed leaks. Its best detection yields a 63.8 % recall and a ten times higher false positive rate.

[4]Rabin's fingerprint is used for unbiased sampling (Sect. 5.4.2).

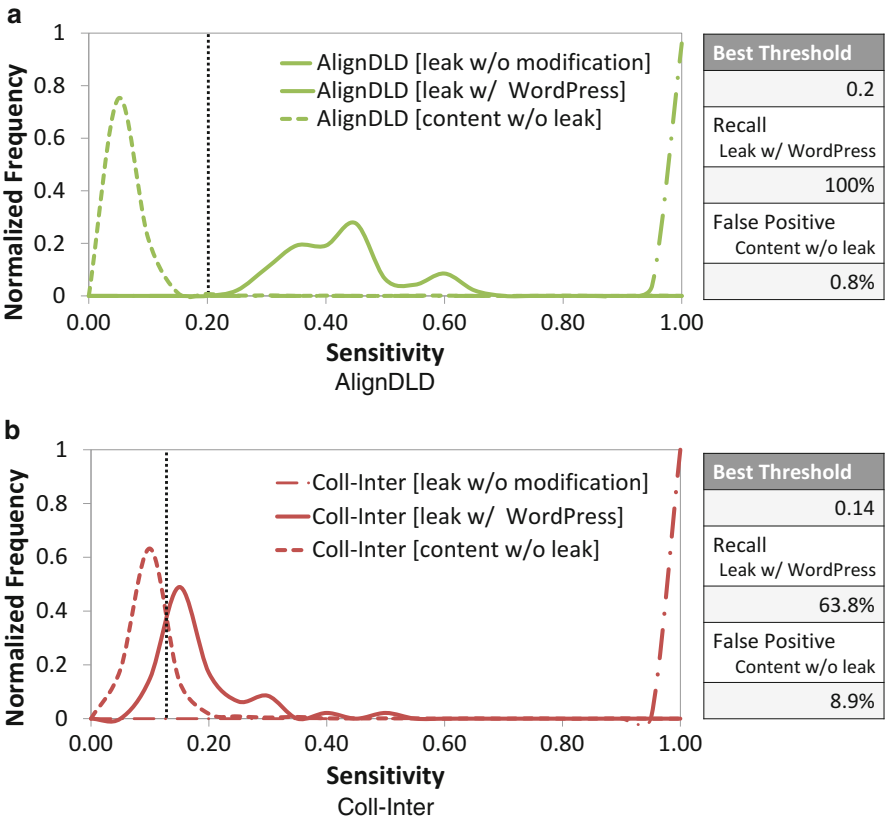

Fig. 5.12 Detection comparison of the leak through `WordPress` using AlignDLD (**a**) and collection intersection (**b**). *Solid lines* show the sensitivity distribution of the modified leaks via `WordPress`

5.4.4.2 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 1 KB. The size of the leaked portion in the content ranges from 32 bytes to 1 KB. Each content sequence is 1 KB long with random padding.

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 (5.5), where $\tilde{\xi}$ is the maximum local alignment score obtained between aligned segments $\tilde{\mathscr{S}}^a$ and $\tilde{\mathscr{S}}^b$, 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 the experiments.

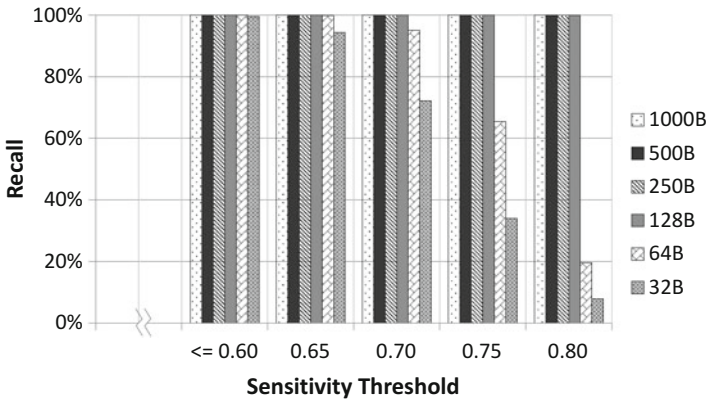

Fig. 5.13 The detection success rate of AlignDLD in partial data leaks

$$\tilde{\mathbb{S}} = \frac{\tilde{\xi}}{r \times \min\left(|\tilde{\mathscr{S}}^a|, |\tilde{\mathscr{S}}^b|\right)} \text{ where } \min\left(|\tilde{\mathscr{S}}^a|, |\tilde{\mathscr{S}}^b|\right) \geq l \tag{5.5}$$

The detection results of AlignDLD are shown in Fig. 5.13. Content with longer sensitive text is easier to detection as expected. Nevertheless, **AlignDLD 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 AlignDLD is resilient to partial data leaks or data truncation.

5.4.5 *Parallelization and Evaluation*

The capability of processing large amounts of content and sensitive data is demonstrated with multithreading CPU and GPU prototypes of AlignDLD.[5]

In the multithreading CPU prototype, both SAMPLING and ALIGNMENT procedures are paralleled with `pthread` library. Long streams are split into multiple substrings, which are sampled in parallel by different threads and then assembled for output. ALIGNMENT is the most time-consuming procedure and is made parallel on both CPU and GPU. A parallelized score matrix filling method is used 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`,

[5]The GPU prototype is realized on one NVIDIA Tesla C2050 with 448 GPU cores.

Table 5.3 Times of speedup in AlignDLD's alignment operation. [P] represents a parallel version

Traffic	*Enron*			*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 5.4 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

`png`, or `pdf` against *Enron* or *MiscNet* (miscellaneous 500 MB network traffic), respectively, are shown in Table 5.3. The GPU AlignDLD prototype achieves over 40 times of speedup over the CPU version on large content datasets. The prototype achieves a throughput of over 400 Mbps against 500 MB misc network traffic (*MiscNet*) shown in Table 5.4. 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.

5.5 Other Defenses Against Data Leaks

In this section, we present other data-leak detection and prevention techniques. Most of them are not designed to analyze leaks in big data. In addition, we introduce the state-of-the-art techniques that could be used in data-leak detection.

5.5.1 Other Data-Leak Detection Techniques

Existing commercial data-leak detection/prevention solutions include Symantec DLP [62], IdentityFinder [28], GlobalVelocity [23], and GoCloudDLP [24]. All solutions are likely based on n-gram set intersection. IdentityFinder searches file systems for short patterns of numbers that may be sensitive (e.g., 16-digit numbers that might be credit card numbers). Symantec DLP is based on n-grams and Bloom filters. The advantage of Bloom filter is space saving. However, filter membership testing is based on unordered n-grams, which generates coincidental matches and false alarms. Most of these commercial detection solutions do not have the privacy-preserving feature and cannot be outsourced. GoCloudDLP [24] is a little different, which allows its customers to outsource the detection to a fully honest DLD provider.

Borders and Prakash [6] presented a network analysis based approach for estimating information leak in the outbound traffic. The method identifies the anomalous and drastic increase in the amount of information carried by the traffic. The method was not designed to detect small-scale data leak. Croft and Caesar [16] compared two logical copies of network traffic to control the movement of sensitive data. The work by Papadimitriou and Garcia-Molina [46] aims at finding the agents that leaked the sensitive data. Blanton et al. [5] proposed a solution for fast outsourcing of sequence edit distance and secure path computation, while preserving the confidentiality of the sequence.

Examples of host-based approaches include Auto-FBI [80] and Aquifer [44]. Auto-FBI guarantees the secure access of sensitive data on the web. It achieves this guarantee by automatically generating a new browser instance for sensitive content. Aquifer is a policy framework and system. It helps prevent accidental information disclosure in OS.

Another approach to the detection of sensitive data leak is to track the data/metadata movement. Several tools are developed for securing sensitive information on mobile platforms [26, 44, 69]. Nadkarni and Enck described an approach to control the sharing of sensitive files among mobile applications [44]. File descriptors (not the content) are stored, tracked and managed. The access control on files is enforced through policies. Yang et al. presented a method aiming at detecting the transmission of sensitive data that is not intended by smartphone users via symbolic execution analysis [69]. Hoyle et al. described a visualization method for informing mobile users of information exposure [26]. The information exposure may be caused by improper setting or configuration of access policies. The visualization is through an avatar apparel approach. Croft and Caesar expand the data tracking from a single host to a network and use shadow packets to distinguish normal traffic from leaks [15]. The security goals and requirements in all these studies are very different from ours, leading to different techniques developed and used.

iLeak is a system for preventing inadvertent information leaks on a personal computer [36]. It takes advantages of the keyword searching utility present in many modern operating systems. iLeak monitors the file access activities of processes and searches for system call inputs that involve sensitive data. Unlike the general data-leak detection approach, iLeak is designed to secure personal data on a single machine, and its detection capability is restricted by the underlying keyword searching utility, which is not designed for detecting either transformed data leaks or partial data leaks.

Bertino and Ghinita addressed the issue of data leaks in the database from the perspective of anomaly detection [2]. Normal user behaviors are monitored and modeled in DBMS, and anomalous activities are identified with respect to potential data-leak activities. Bertino also discussed watermarking and provenance techniques used in data-leak prevention and forensics [2], which is investigated in details by Papadimitriou and Garcia-Molina in [46].

5.5.2 Existing Privacy-Preserving Techniques

There have been several advances in understanding the privacy needs [37] or the privacy requirement of security applications [66]. In this chapter, 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 [50] was used previously for identifying similar spam messages in a collaborative setting [40], as well as collaborative worm containment [10], virus scan [25], and fragment detection [52].

There are also other privacy-preserving techniques invented for specific processes, e.g., DNA matching [63], or for general purpose use, e.g., secure multi-party computation (SMC). Similar to string matching methods discussed above, [63] uses anonymous automata to perform the comparison. Shu and Yao presented privacy-preserving methods for protecting sensitive data in a non-MapReduce based detection environment [57]. SMC [70] 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 [71], automated troubleshooting [27], network event statistics [9, 65], private information retrieval [74], genomic computation [32], private database query [73], private join operations [11], and distributed data mining [29]. The provable privacy guarantees offered by SMC comes at a cost in terms of computational complexity and realization difficulty.

5.5.3 Applications and Improvements of MapReduce

MapReduce framework was used to solve problems in data mining [78], machine learning [45], database [4, 64], and bioinformatics [43]. MapReduce algorithms for computing document similarity (e.g., [1, 18, 76]) involve pairwise similarity comparison. Similarity measures may be Hamming distance, edit distance or Jaccard distance. MapReduce was also used by security applications, such as log analysis [22, 68], spam filtering [12, 13] and malware and botnet detection [21, 49, 79] for scalability. The security solutions proposed by Bilge et al. [3], Yang et al. [68] and Yen et al. [72] analyzed network traffic or logs with MapReduce, searching for malware signatures or behavior patterns. None of the existing MapReduce solutions addresses data-leak detection problem.

Several techniques have been proposed to improve the privacy protection of MapReduce framework. Such solutions typically assume that the cloud provider is trustworthy. For example, Pappas et al. [47] proposed a data-flow tracking method in cloud applications. It audits the use of the sensitive data sent to the cloud. Roy et al. [54] integrate mandatory access control with differential privacy in order to manage the use of sensitive data in MapReduce computations. Yoon and Squicciarini [75] detected malicious or cheating MapReduce nodes by correlating different

nodes' system and Hadoop logs. Squicciarini et al. [60] presented techniques that prevent information leakage from the indexes of data in the cloud.

There exist MapReduce algorithms for computing the set intersection [4, 64]. They differ from the collection intersection algorithms, as explained in Sect. 5.2.2. Collection intersection algorithm requires new intermediate data fields and processing for counting and recording duplicates in the intersection. Several techniques were developed for monitoring or improving MapReduce performance, e.g., to identify nodes with slow tasks [14], GPU acceleration [19] and efficient data transfer [42]. These advanced techniques can be applied to further speed up MR-DLD.

5.6 Conclusions

In this chapter, we introduced the background of data-leak detection and presented two techniques in detail, both of which are able to rapidly screen big content for inadvertent data exposure.

MR-DLD is a MapReduce based system for detecting the occurrences of sensitive data patterns in massive-scale content in data storage or network transmission. It provides privacy enhancement to minimize the exposure of sensitive data during the outsourced detection. MR-DLD was also deployed and evaluated with the Hadoop platform on Amazon EC2 and a local cluster, and it achieved 225 Mbps analysis throughput.

AlignDLD is based on aligning two sampled sequences for similarity comparison. The main feature of this solution is its ability to detect transformed or modified leaks. This approach consists of a comparable sampling algorithm and a specialized alignment algorithm for comparing two sampled sequences. The unique design of the two algorithms enables its accuracy and scalability. The extensive experimental evaluation with real-world data and leak scenarios confirms that this method has a high specificity (i.e., low false alarm rate) and detects transformed data leaks much more effectively than collection-intersection methods.

5.7 Future Work

The MR-DLD approach could be improved by being deployed on hybrid cloud environments, which consist of private machines owned by the data owner and public machines owned by the cloud provider. It is another general approach that secures computation with mixed sensitive data. The MapReduce system in the hybrid cloud is installed across public cloud and private cloud (e.g., [77]). It may require much more computation capacity for data owner when compared with our specific architecture. The use of hybrid cloud infrastructure will likely improve the efficiency of the detection system.

Alignment algorithm on MapReduce has been studied in bioinformatics for DNA sequence read mapping (e.g., [55, 56]). The AlignDLD approach could also be implemented on MapReduce, improving its privacy-preserving ability and scalability. Each mapper and reducer could be sped up with the MapReduce on GPU cluster framework (e.g., [61]). By combining the AlignDLD approach and the MR-DLD approach with these latest techniques, future data-leak detection approaches can achieve higher detection accuracy and scalability.

Acknowledgements This work has been supported by NSF S^{2}ERC Center (an I/UCRC Center) and ARO YIP grant W911NF-14-1-0535.

References

1. Baraglia R, Morales GDF, Lucchese C (2010) Document similarity self-join with MapReduce. In: 2010 IEEE 10th international conference on data mining (ICDM). IEEE Computer Society, Sydney, Australia, pp 731–736
2. Bertino E, Ghinita G (2011) Towards mechanisms for detection and prevention of data exfiltration by insiders: keynote talk paper. In: Proceedings of the 6th ACM symposium on information, computer and communications security, ASIACCS '11, pp 10–19
3. Bilge L, Balzarotti D, Robertson W, Kirda E, Kruegel C (2012) Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In: Proceedings of the 28th annual computer security applications conference, ACSAC '12. ACM, New York, NY, pp 129–138. doi:10.1145/2420950.2420969. http://doi.acm.org/10.1145/2420950.2420969
4. Blanas S, Patel JM, Ercegovac V, Rao J, Shekita EJ, Tian Y (2010) A comparison of join algorithms for log processing in MapReduce. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data, SIGMOD '10. ACM, New York, NY, pp 975–986 doi:10.1145/1807167.1807273. http://doi.acm.org/10.1145/1807167.1807273
5. Blanton M, Atallah MJ, Frikken KB, Malluhi QM (2012) Secure and efficient outsourcing of sequence comparisons. In: Computer security - ESORICS 2012 - 17th European symposium on research in computer security, Proceedings, Pisa, 10–12 Sept 2012, pp 505–522. doi:10.1007/978-3-642-33167-1_29. http://dx.doi.org/10.1007/978-3-642-33167-1_29
6. Borders K, Prakash A (2009) Quantifying information leaks in outbound web traffic. In: IEEE symposium on security and privacy. IEEE Computer Society, San Jose, CA, USA, pp 129–140
7. Broder AZ (1993) Some applications of Rabin's fingerprinting method. In: Capocelli R, De Santis A, Vaccaro U (eds) Sequences II. Springer, New York, pp 143–152. doi:10.1007/978-1-4613-9323-8_11. http://dx.doi.org/10.1007/978-1-4613-9323-8_11
8. Broder AZ (2000) Identifying and filtering near-duplicate documents. In: Proceedings of the 11th annual symposium on combinatorial pattern matching, pp 1–10
9. Burkhart M, Strasser M, Many D, Dimitropoulos X (2010) Sepia: privacy-preserving aggregation of multi-domain network events and statistics. In: Proceedings of the 19th USENIX Security Symposium, pp 15–15
10. Cai M, Hwang K, Kwok YK, Song S, Chen Y (2005) Collaborative Internet worm containment. IEEE Secur Priv 3(3):25–33
11. Carbunar B, Sion R (2010) Joining privately on outsourced data. In: Secure data management. Lecture notes in computer science, vol 6358. Springer, Berlin, pp 70–86
12. Caruana G, Li M, Qi, H (2010) SpamCloud: a MapReduce based anti-spam architecture. In: Seventh international conference on fuzzy systems and knowledge discovery. IEEE, Yantai, Shandong, China, pp 3003–3006

13. Caruana G, Li M, Qi M (2011) A MapReduce based parallel SVM for large scale spam filtering. In: Eighth international conference on fuzzy systems and knowledge discovery. IEEE, Shanghai, China, pp 2659–2662
14. Chen Q, Liu C, Xiao Z (2014) Improving MapReduce performance using smart speculative execution strategy. IEEE Trans Comput 63(4):954–967. doi:10.1109/TC.2013.15
15. Croft J, Caesar M (2011) Towards practical avoidance of information leakage in enterprise networks. In: Proceedings of the 6th USENIX conference on hot topics in security, HotSec'11, pp 7–7
16. Croft J, Caesar, M (2011) Towards practical avoidance of information leakage in enterprise networks. In: 6th USENIX workshop on hot topics in security, HotSec'11. USENIX Association
17. Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
18. Elsayed T, Lin JJ, Oard DW (2008) Pairwise document similarity in large collections with MapReduce. In: ACL (Short Papers). The Association for Computer Linguistics, pp 265–268
19. Fang W, He B, Luo Q, Govindaraju NK (2011) Mars: accelerating MapReduce with graphics processors. IEEE Trans Parallel Distrib Syst 22(4):608–620
20. FBI Cyber Division (2014) Recent cyber intrusion events directed toward retail firms
21. François J, Wang S, Bronzi W, State R, Engel T (2011) BotCloud: detecting botnets using MapReduce. In: IEEE international workshop on information forensics and security. IEEE, Iguacu Falls, Brazil, pp 1–6
22. Fu X, Ren R, Zhan J, Zhou W, Jia Z, Lu G (2012) LogMaster: mining event correlations in logs of large-scale cluster systems. In: IEEE 31st symposium on reliable distributed systems. IEEE, Irvine, CA, USA, pp 71–80
23. Global Velocity Inc (2015) Global velocity inc. http://www.globalvelocity.com/. Accessed Feb 2015
24. GTB Technologies Inc (2015) GoCloudDLP. http://www.gocloudlp.com/. Accessed Feb 2015
25. Hao F, Kodialam M, Lakshman T, Zhang H (2005) Fast payload-based flow estimation for traffic monitoring and network security. In: Proceedings of the 2005 symposium on architecture for networking and communications systems, pp 211–220
26. Hoyle R, Patil S, White D, Dawson J, Whalen P, Kapadia A (2013) Attire: conveying information exposure through avatar apparel. In: Proceedings of the 2013 conference on computer supported cooperative work companion, CSCW '13, pp 19–22
27. Huang Q, Jao D, Wang HJ (2005) Applications of secure electronic voting to automated privacy-preserving troubleshooting. In: Proceedings of the 12th ACM conference on computer and communications security, pp 68–80
28. Identifyfinder (2015) Identity finder. http://www.identityfinder.com/. Accessed Feb 2015
29. Jagannathan G, Wright RN (2005) Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In: Proceedings of the 11th ACM SIGKDD international conference on knowledge discovery in data mining, pp 593–599
30. Jang J, Brumley D, Venkataraman S (2011) BitShred: feature hashing malware for scalable triage and semantic analysis. In: Proceedings of the 18th ACM conference on computer and communications security, CCS '11, pp 309–320
31. Jang Y, Chung S, Payne B, Lee W (2014) Gyrus: a framework for user-intent monitoring of text-based networked applications. In: Proceedings of the 23rd USENIX security symposium, pp 79–93
32. Jha S, Kruger L, Shmatikov V (2008) Towards practical privacy for genomic computation. In: Proceedings of the 29th Ieee symposium on security and privacy, pp 216–230
33. Jung J, Sheth A, Greenstein B, Wetherall D, Maganis G, Kohno T (2008) Privacy oracle: a system for finding application leaks with black box differential testing. In: Proceedings of the 15th ACM conference on computer and communications security, pp 279–288
34. Kalyan C, Chandrasekaran K (2007) 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, pp 104–109

35. Kaspersky Lab (2014) Kaspersky lab IT security risks survey 2014: a business approach to managing data security threats
36. Kemerlis VP, Pappas V, Portokalidis G, Keromytis AD (2010) iLeak: a lightweight system for detecting inadvertent information leaks. In: Proceedings of the 6th European conference on computer network defense
37. Kleinberg J, Papadimitriou CH, Raghavan P (2001) On the value of private information. In: Proceedings of the 8th conference on theoretical aspects of rationality and knowledge, pp 249–257
38. Lam W, Liu L, Prasad S, Rajaraman A, Vacheri Z, Doan A (2012) Muppet: Mapreduce-style processing of fast data. Proc VLDB Endow 5(12):1814–1825. doi:10.14778/2367502.2367520. http://dx.doi.org/10.14778/2367502.2367520
39. Lee Y, Kang W, Son H (2010) An internet traffic analysis method with MapReduce. In: Network operations and management symposium workshops (NOMS Wksps), 2010 IEEE/IFIP, pp 357–361. doi:10.1109/NOMSW.2010.5486551
40. Li K, Zhong Z, Ramaswamy L (2009) Privacy-aware collaborative spam filtering. IEEE Trans Parallel Distrib Syst 20(5):725–739
41. Liu F, Shu X, Yao D, Butt AR (2015) Privacy-preserving scanning of big content for sensitive data exposure with mapreduce. In: Proceedings of the 5th ACM conference on data and application security and privacy, CODASPY 2015, San Antonio, TX, 2–4 Mar 2015, pp 195–206
42. Logothetis D, Trezzo C, Webb KC, Yocum K (2011) In-situ MapReduce for log processing. In: USENIX annual technical conference. USENIX Association
43. Matsunaga AM, Tsugawa MO, Fortes JAB (2008) Cloudblast: combining MapReduce and virtualization on distributed resources for bioinformatics applications. In: eScience. IEEE Computer Society, Indianapolis, IN, USA, pp 222–229
44. Nadkarni A, Enck W (2013) Preventing accidental data disclosure in modern operating systems. In: ACM conference on computer and communications security. ACM, Berlin, Germany, pp 1029–1042
45. Panda B, Herbach JS, Basu S, Bayardo RJ (2009) Planet: massively parallel learning of tree ensembles with MapReduce. Proc VLDB Endow 2(2):1426–1437. doi:10.14778/1687553.1687569. http://dx.doi.org/10.14778/1687553.1687569
46. Papadimitriou P, Garcia-Molina H (2011) Data leakage detection. IEEE Trans Knowl Data Eng 23(1):51–63
47. Pappas V, Kemerlis V, Zavou A, Polychronakis M, Keromytis A (2013) Cloudfence: enabling users to audit the use of their cloud-resident data. In: Research in attacks, intrusions, and defenses. Lecture notes in computer science, vol 8145. Springer, Berlin, pp 411–431. doi:10.1007/978-3-642-41284-4_21. http://dx.doi.org/10.1007/978-3-642-41284-4_21
48. Peng D, Dabek F (2010) Large-scale incremental processing using distributed transactions and notifications. In: Proceedings of the 9th USENIX conference on operating systems design and implementation, OSDI'10. USENIX Association, Berkeley, CA, pp 1–15. http://dl.acm.org/citation.cfm?id=1924943.1924961
49. Provos N, McNamee D, Mavrommatis P, Wang K, Modadugu N (2007) The ghost in the browser: analysis of web-based malware. In: First workshop on hot topics in understanding botnets. USENIX Association
50. Rabin MO (1981) Fingerprinting by random polynomials. Technical Report TR-15-81, The Hebrew University of Jerusalem
51. Rabin MO (1981) Fingerprinting by random polynomials. Technical Report TR-15-81, Harvard Aliken Computation Laboratory
52. Ramaswamy L, Iyengar A, Liu L, Douglis F (2004) Automatic detection of fragments in dynamically generated web pages. In: Proceedings of the 13th international conference on world wide web, pp 443–454
53. RiskBasedSecurity (2015) Data breach quickview: 2014 data breach trends
54. Roy I, Setty STV, Kilzer A, Shmatikov V, Witchel E (2010) Airavat: security and privacy for MapReduce. In: Proceedings of the 7th USENIX symposium on networked systems design and implementation, pp 297–312. USENIX Association

55. Schatz MC (2008) Blastreduce: high performance short read mapping with mapreduce. University of Maryland. http://cgis.cs.umd.edu/Grad/scholarlypapers/papers/MichaelSchatz.pdf
56. Schatz MC (2009) Cloudburst: highly sensitive read mapping with mapreduce. Bioinformatics 25(11):1363–1369. doi:10.1093/bioinformatics/btp236. http://bioinformatics.oxfordjournals.org/content/25/11/1363.abstract
57. Shu X, Yao D (2012) Data leak detection as a service. In: Proceedings of the 8th international conference on security and privacy in communication networks (SecureComm), Padua, pp 222–240
58. Shu X, Zhang J, Yao D, Feng W (2015) Rapid and parallel content screening for detecting transformed data exposure. In: Proceedings of the third international workshop on security and privacy in big data (BigSecurity). Hongkong, China
59. Smith TF, Waterman MS (1981) Identification of common molecular subsequences. J Mol Biol 147(1):195–197
60. Squicciarini AC, Sundareswaran S, Lin D (2010) Preventing information leakage from indexing in the cloud. In: IEEE international conference on cloud computing, CLOUD 2010, Miami, FL 5–10 July. IEEE, Miami, FL, USA, pp 188–195. doi:10.1109/CLOUD.2010.82. http://dx.doi.org/10.1109/CLOUD.2010.82
61. Stuart JA, Owens JD (2011) Multi-gpu mapreduce on gpu clusters. In: Proceedings of the 2011 ieee international parallel & distributed processing symposium, IPDPS '11. IEEE Computer Society, Washington, DC, pp 1068–1079. doi:10.1109/IPDPS.2011.102. http://dx.doi.org/10.1109/IPDPS.2011.102
62. Symantec (2015) Symantec data loss prevention. http://www.symantec.com/data-loss-prevention. Accessed Feb 2015
63. Troncoso-Pastoriza JR, Katzenbeisser S, Celik M (2007) Privacy preserving error resilient DNA searching through oblivious automata. In: Proceedings of the 14th ACM conference on computer and communications security, pp 519–528
64. Vernica R, Carey MJ, Li C (2010) Efficient parallel set-similarity joins using MapReduce. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data, SIGMOD '10. ACM, New York, NY, pp 495–506. doi:10.1145/1807167.1807222. http://doi.acm.org/10.1145/1807167.1807222
65. Williams P, Sion R (2008) Usable PIR. In: Proceedings of the 13th network and distributed system security symposium
66. Xu S (2009) 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. Springer, Berlin, pp 217–228
67. Xu K, Yao D, Ma Q, Crowell A (2011) Detecting infection onset with behavior-based policies. In: Proceedings of the 5th international conference on network and system security, pp 57–64
68. Yang SF, Chen WY, Wang YT (2011) ICAS: an inter-VM IDS log cloud analysis system. In: 2011 IEEE international conference on cloud computing and intelligence systems (CCIS), pp 285–289. doi:10.1109/CCIS.2011.6045076
69. Yang Z, Yang M, Zhang Y, Gu G, Ning P, Wang XS (2013) AppIntent: analyzing sensitive data transmission in Android for privacy leakage detection. In: Proceedings of the 20th ACM conference on computer and communications security
70. Yao ACC (1986) How to generate and exchange secrets. In: Proceedings of the 27th annual symposium on foundations of computer science, pp 162–167
71. Yao D, Frikken KB, Atallah MJ, Tamassia R (2008) Private information: to reveal or not to reveal. ACM Trans Inf Syst Secur 12(1):6
72. Yen TF, Oprea A, Onarlioglu K, Leetham T, Robertson W, Juels A, Kirda E (2013) Beehive: large-scale log analysis for detecting suspicious activity in enterprise networks. In: Proceedings of the 29th annual computer security applications conference, ACSAC '13. ACM, New York, pp 199–208. doi:10.1145/2523649.2523670. http://doi.acm.org/10.1145/2523649.2523670

73. Yi X, Kaosar MG, Paulet R, Bertino E (2013) Single-database private information retrieval from fully homomorphic encryption. IEEE Trans Knowl Data Eng 25(5):1125–1134
74. Yi X, Paulet R, Bertino E (2013) Private information retrieval. Synthesis lectures on information security, privacy, and trust. Morgan & Claypool Publishers
75. Yoon E, Squicciarini A (2014) Toward detecting compromised mapreduce workers through log analysis. In: 2014 14th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGrid), pp 41–50. doi:10.1109/CCGrid.2014.120
76. Yuan P, Sha C, Wang X, Yang B, Zhou A, Yang S (2010) XML structural similarity search using MapReduce. In: 11th international conference, web-age information management. Lecture notes in computer science, vol 6184. Springer, New York, pp 169–181
77. Zhang C, Chang EC, Yap R (2014) Tagged-MapReduce: a general framework for secure computing with mixed-sensitivity data on hybrid clouds. In: 2014 14th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGrid), pp 31–40. doi:10.1109/CCGrid.2014.96
78. Zhao W, Ma H, He Q (2009) Parallel *k*-means clustering based on MapReduce. In: Cloud computing, first international conference, CloudCom 2009. lecture notes in computer science, vol 5931. Springer, Berlin. pp 674–679
79. Zhuang L, Dunagan J, Simon DR, Wang HJ, Osipkov I, Tygar JD (2008) Characterizing botnets from Email spam records. In: First USENIX workshop on large-scale exploits and emergent threats, LEET '08. USENIX Association
80. Zohrevandi M, Bazzi RA (2013) Auto-FBI: a user-friendly approach for secure access to sensitive content on the web. In: Proceedings of the 29th annual computer security applications conference, ACSAC '13. ACM, New York, NY, pp 349–358. doi:10.1145/2523649.2523683. http://doi.acm.org/10.1145/2523649.2523683