IRILD: an Information Retrieval based method for Information Leak Detection

Eleni Gessiou [1], Quang Hieu Vu [2], Sotiris Ioannidis [1]
[1] Institute of Computer Science, FORTH, Greece
[2] Etisalat BT Innovation Center, Khalifa University, UAE

***Abstract*—The traditional approach for detecting information leaks is to generate fingerprints of sensitive data, by partitioning and hashing it, and then comparing these fingerprints against outgoing documents. Unfortunately, this approach incurs a high computation cost as every part of document needs to be checked. As a result, it is not applicable to systems with a large number of documents that need to be protected. Additionally, the approach is prone to false positives if the fingerprints are common phrases. In this paper, we propose an improvement for this approach to offer a much faster processing time with less false positives. The core idea of our solution is to eliminate common phrases and non-sensitive phrases from the fingerprinting process. Non-sensitive phrases are identified by looking at available public documents of the organization that we want to protect from information leaks and common phrases are identified with the help of a search engine. In this way, our solution both *accelerates* leak detection and *increases* the accuracy of the result. Experiments were conducted on real-world data to prove the efficiency and effectiveness of the proposed solution.**

***Keywords*-privacy, information leaks, fingerprinting**

I. Introduction

Information leaks are a critical problem of computer systems. The leak of confidential data, be it accidental or intentional, may cause huge losses to the data owner. These losses may not be only financial, such as termination of contracts or compensation for customers, but also reputation loss whose cost cannot be estimated easily. In particular, according to a study [1] in 2006 conducted by the Ponemon Institute at 31 organizations that lost confidential information, the average cost of a case of information leak was approximately 4.8 million USD. Typical examples of information leak include the case of MediaDefender in 2007 when over 6,000 internal emails were leaked to the outside world [1] and the case of ACS:Law firm in September of 2010 when personal details of 5,300 people became public [2].

There are two primary solutions for information leak detection. The first one is to use specific expressions, keywords or phrases to identify confidential information. For example, a leak of a Mastercard number can be detected by searching expressions of 16 digits starting with two digits in the range from 51 to 55. While this solution is simple and easily applied, its main disadvantage is that it cannot be employed if confidential information cannot be well-defined by expressions, keywords, or phrases. In this case, the alternative solution is to generate fingerprints of confidential information, which can appear in any structure, and check the generated fingerprints against fingerprints obtained from outgoing traffic.

To protect a confidential document from information leaks, a simple approach is to generate a fingerprint of the whole document and check this fingerprint against fingerprints of all outgoing documents. Since this approach cannot handle the case where the leaked information is only in part of the document, the popular approach is to employ cyclical hashing to generate a series of fingerprints for the document and use this series of fingerprints, which in turn are checked against the series of fingerprints of outgoing documents. Nevertheless, there are still two weaknesses in this popular approach. First, the popular approach incurs a high cost in both fingerprint generation and leak detection since every part of a document needs to be checked. Consequently, due to the high leak detection cost, this approach is not applicable to systems where a large number of documents must be protected. Second, it is prone to false positives if a lot of common phrases are used in confidential documents, and expectedly this will create a lot of hits when checked against outgoing documents. The reason is because these phrases often exist in all documents.

To address the problems of the popular approach, we propose a solution based on information retrieval to identify only phrases containing sensitive information for fingerprinting. The basic idea of our solution is to check the popularity of phrases before fingerprinting in two ways. We first look at available public documents of the company or organization that we want to protect from information leak. If the phrase exists in these documents, it does not convey any secret information, and hence it is not a sensitive phrase. We then submit the phrase to a search engine such as Google and measure the number of returned results. Intuitively, the higher the number of

[1] http://www.usatoday.com/tech/news/computersecurity/hacking/2007-09-18-mediadefender-leak_N.htm?csp=34
[2] http://www.bbc.co.uk/news/technology-11418962

returned results is, the more popular the phrase is. What this means is that if a phrase has a large number of returned results, it is a common phrase. By removing public and common phrases in documents from the fingerprint generation process, our solution reduces both the cost of fingerprint generation and leak detection, offering higher processing speed. Furthermore, our solution can improve the accuracy of detection by reducing false positives caused by public and common phrases. In summary, our work makes the following major contributions:

- We propose a novel solution to improve the performance of the traditional approach for information leak detection in terms of processing speed and accuracy. Our core idea is to identify non-sensitive phrases as well as common phrases, and eliminating them from the fingerprinting process of confidential documents.
- To evaluate the popularity of a long combined phrase which has not returned results from search engines, we propose a novel technique to split the phrase into sub-phrases and identify the popularity of the phrase based on the popularity of its divided phrases.
- We conducted an extensive experimental evaluation of the effectiveness and efficiency of our proposed method.

The rest of the paper is organized as follows. In Sections II and III, we discuss prior work and introduce background knowledge. In Sections IV and V, we present and evaluate our solution. Finally, we conclude in Section VI.

II. Related Work

In the past few years, a number of white papers have been written discussing different aspects of information leak prevention in general, and how to detect information leak in particular. These include [2], which introduces basic solutions to detect and prevent information leaks, [3], which presents testing and evaluation standards for information leak prevention products and [1], which studies the cost incurred by information leaks in practice.

In general, as discussed in [2], there are two main approaches to information leak detection. One is based on defining sensitive expressions, keywords or phrases. This way, information leaks are detected if the outgoing traffic contains the specified expressions, keywords or phrases. The other is based on fingerprints of information. For example, a popular approach for information leak detection in documents is to divide them into multiple parts and generate fingerprints of these parts. These fingerprints are checked against fingerprints of similar divided parts of outgoing traffic for leak detection.

A special type of information leak is information leak from applications. To deal with this type of leak, [4] proposes Privacy Oracle, a solution that tests an application with different inputs and maps input perturbations to output perturbations to detect potential information leaks. Alternatively, [5] introduces the use of shadow execution that runs two copies of an application at the same time in which, the one containing personal information is kept away from accessing the network while the other with non-confidential data is used to communicate over the network. The response from the network is then shared for both copies. These solutions are, in fact, complementary to these basic solutions as well as the solution presented in this paper.

In addition to information leak detection, there exists a class of techniques that address access control to prevent information leak. Access control is required in cases where the information is available to someone but should be restricted from others. With respect to this aspect, [6], [7], [8], [9] introduce solutions to avoid information leaks caused by accidentally sending emails to unintended recipients. The basic idea of these solutions is to perform data analysis to measure the similarity between the current outgoing email and previous outgoing emails of the same recipient. If there is a big difference between them, the current outgoing email may contain information leak. On the other hand, [10] presents CLAMP, an architecture that protects confidential information in web servers by enforcing strong access control on user data while isolating code running on behalf of different users. These access control techniques are orthogonal to the basic solutions as well as the solution presented in this paper.

The use of search engines to detect information leaks has been introduced in [11]. However, in our work, the purpose of using search engines is simply to detect inferences between keywords. The purpose of finding inferences between keywords is to discover sensitive documents that do not contain specified sensitive keywords but contain closely associated keywords. It is because with high probability, these sensitive documents also contain confidential data. With respect to inference detection, prior to this work, web based inference has been significantly studied in a number of papers such as [12], [13], [14].

III. Background

A popular approach to detect information leaking from a confidential document is to employ cyclical hashing to split the document into multiple parts and generate fingerprints for these parts. In particular, given a document, the method repeatedly creates fingerprints for strings of C characters from the start to the end of the document, offset by O characters each time (C and O are predefined parameters). An example is shown in Figure 1, where C and O are set to 30 and 10 respectively. In this example, 30 characters from the 1^{st} to the 30^{th} positions are used to generate fingerprint1, 30 characters from the 10^{th} to

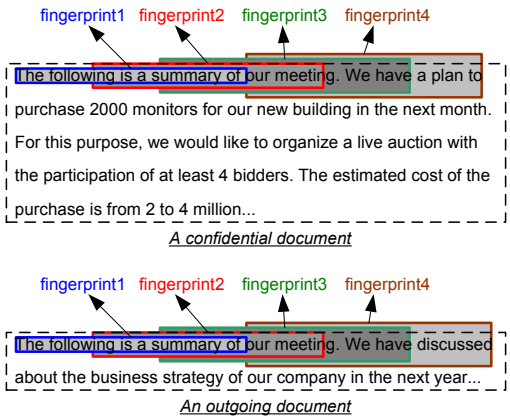


Figure 1. An example of using cyclical hashing

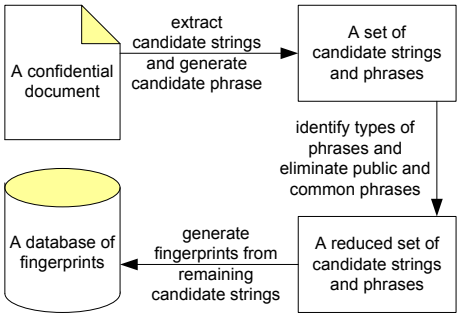


Figure 2. An overview of fingerprint generation

the 40^{th} positions are used to generate fingerprint2, 30 characters from the 20^{th} to the 50^{th} positions are used to generate fingerprint3, and so on.

Given an outgoing traffic channel (e.g., an outgoing email or a file uploading to an outside server), cyclical hashing is also employed to generate a set of fingerprints for the traffic. These fingerprints are then checked against those previously extracted from confidential documents to detect information leaks. In the example in Figure 1, since the first three fingerprints of the outgoing document match the first three fingerprints of the confidential document, it is considered to have partial information leak.

A problem with this approach, illustrated in Figure 1, is that it introduces false positives when common phrases or sentences are used. In this example, even though the common sentence "the following is a summary of our meeting" appears in both the confidential document and the outgoing document, since it does not convey sensitive information, there is actually no information leakage. Furthermore, this redundant check incurs a high processing cost, and hence the approach fails to work in systems with a large number of sensitive documents.

IV. IRILD

To avoid false positives involving common phrases as shown in the example of Figure 1 and also reduce the unnecessary cost of generating and checking fingerprints of the common phrases, we propose IRILD, an information retrieval based method that is able to identify common phrases and eliminate them from the fingerprinting process. In our method, we evaluate the popularity of phrases by submitting them to an Internet search engine such as Google[3] and measure the number of returned results. The higher the number of returned results of a phrase is, the more common the phrase is. For example, since there are approximately 15,300 results returned from Google when searching for the sentence "the following is a summary of our meeting", the sentence is considered as a common sentence and no fingerprint should be generated for it. Furthermore, assume that the organization, which employs IRILD also maintains public documents (e.g., documents in public folders of the company's web site). IRILD will also eliminate phrases that can be found in those public documents from the fingerprinting process because these phrases contain already known information.

IRILD generates fingerprints for confidential documents in three steps. In the first step, similar to the popular approach introduced in Section III, IRILD employs cyclical hashing on each confidential document to generate a set of candidate strings for the fingerprinting process. Note that by using a fixed number of characters to generate strings, a candidate string may not be a complete phrase (e.g., as in the example of Figure 1, the second candidate string, "ing is a summary of our meetin" is not a complete phrase). Thus, from this set of candidate strings, IRILD needs to generate a set of candidate, complete phrases. Each phrase corresponds to a candidate string and is the shortest phrase that totally covers the string. For example, the candidate phrase corresponding to the second candidate string in the example of Figure 1 is "following is a summary of our meeting". In the second step, IRILD identifies public phrases and common phrases and removes them from the set of candidate phrases. Finally, candidate strings associated with remaining sensitive candidate phrases are used to generate fingerprints. The overview of IRILD's processing steps to generate fingerprints for a confidential document is illustrated in Figure 2.

Note that while the fingerprint generation of IRILD is different from that of the popular approach, the information leak detection of these two approaches is still the same, i.e., fingerprints of confidential documents in the database are used to check against fingerprints of outgoing documents for information leak detection. Also, note that both cyclical hashing method and IRILD do not work for encoded, encrypted, or compressed data. The aim of cyclic hashing method and IRILD is to prevent information

[3]http://www.google.com

leak in normal text information channels. We believe that solutions to deal with concealed information (e.g., encoded or encrypted data) and regular expression patterns or time aspect of the information are complementary to IRILD.

A. Public phrase identification

The task of identifying public phrases from a set of candidate phrases is simply a search of these phrases from available public documents. To fulfill the task, a solution is to employ the basic information retrieval technique to create document indices for public documents and inverted indices for words in public documents. Each document index records words that appear in a public document. On the other hand, each inverted index records positions of a word in all documents the word appears. For example, the structure of a document index can be $\{doc_x$: $word_1$, $word_2$, $word_3$, $\cdots\}$ while the structure of an inverted index can be $\{word_y$: $[doc_1$: pos_{11}, pos_{12}, pos_{13}, $\cdots]$, $[doc_2$: pos_{21}, pos_{22}, pos_{23}, $\cdots]$, $\cdots\}$. To detect whether a phrase is a public phrase, we first parse the phrase into a list of words. After that, we search document indices to see if there is any document that contains all words in the list. If such a document exists, we then retrieve inverted indices of the words to see their positions in the document. If the words appear at adjacent positions, it forms a phrase in the document. In this case, we conclude the checking phrase is a public phrase since the checking phrase matches a phrase in a public document. While this solution always generates exact results without false positives, it incurs a high cost in search. As a result, this technique can only be used if the number of public documents is not very large. An alternative solution is to also employ cyclical hashing to generate fingerprints for public documents and compare these fingerprints to the fingerprints of the candidate strings to check if the candidate strings exist in public documents. This technique is often used if the number of public documents is large.

B. Common phrase identification

As previously discussed, to check whether a phrase is a common phrase, we submit the phrase to Google and measure the number of returned results. Basically, a phrase is a common phrase if there are a lot of returned results. As discussed before if the phrase has a lot of returned results from the search engine, we hypothesize that there is a high probability that it does not contain secret information. On the other hand, if the search engine returns only a few results for the search, the phrase is considered as a sensitive phrase. To make it easy to evaluate the popularity of a phrase, we propose a formula to calculate the popularity score of the phrase from its number of returned results by the search engine as follows:

$$Score(P) = \log_{10}(N(P) + 1) \quad (1)$$

where P is the evaluating phrase and $N(P)$ is the number of results returned from searching P. A common phrase is a phrase whose popularity score is greater than a predefined threshold K. For example, if we set K to 4 and the search of a phrase P has 20,000 number of results, P is considered as a common phrase because $Score(P) = \log_{10}(20,000 + 1) > 4$.

The above technique to identify common phrases usually works if candidate phrases are not long (e.g., common phrases are single phrases). However, in cases where candidate phrases are long (e.g., when C is set to a large value, the candidate phrases that cover candidate strings are also long), a problem comes. In these cases, candidate phrases can be a combined phrase, a sentence, or even some sentences or a paragraph. While it is easy to identify the popularity of a single phrase by the search engine, it is more difficult to do the same thing for a combined long phrase or sentence. It is because with high probability, the long phrase cannot be found by the search engine. To deal with this problem, we suggest a simple way to parse the combined phrase or sentence into smaller single phrases and calculate the popularity of the combined phrase from the popularity of the split single phrases. In particular, let P_1, P_2, $\cdots$, P_n be n single phrases that are split from a combined phrase P and $Score(P_1)$, $Score(P_2)$, $\cdots$, $Score(P_n)$ be their popularity scores. $Score(P)$ is calculated as:

$$Score(P) = \min_{i=1..n} \{Score(P_i)\} \quad (2)$$

The rationale behind the above formula is that the score of the combined phrase should be equal to the minimum popularity score of its members. The intuition of the formula is straightforward. A combined phrase is a common phrase if all of its sub-phrases are common. On the other hand, a combined phrase is a sensitive phrase if at least one of its split phrases is a sensitive phrase.

A concern with this approach is that sensitive information may be leaked if a determined adversary can capture all queries submitted to the search engine and reconstruct indexing documents from these queries. Our solution to this concern is to submit queries from different locations to hide the fact that all queries come from the same source, and hence the attack should fail. In particular, in our experiments, we distributed queries to PlanetLab nodes [15] in different countries across continents and only submitted the queries to the search engine from there. Note that alternative to PlanetLab nodes, search proxies [16], [17] can be used to anonymize the queries. Furthermore, besides Google, we also employed other search engines such as Yahoo![4] and Bing[5] for query submission. By employing multiple sources to send the

[4]http://www.yahoo.com
[5]http://www.bing.com

queries and multiple destinations (i.e., search engines) to process the queries, we have created "noise" for our method. In particular, the noise can be adjusted by increasing or decreasing the number of sources and destination according to the following formula:

$$N(PlanetLab\_Nodes) * N(Search\_Engines) \quad (3)$$

where $N(PlanetLab\_Nodes)$ is the number of PlanetLab nodes and $N(Search\_Engines)$ is the number of search engines that we used in the experiments. With the use of noise, it will require an extremely high cost to inspect all PlanetLab nodes and search engines if an adversary wants to reconstruct indexing documents from the submitted queries.

C. Improvement techniques

Besides the basic solution to identify public phrase, we suggest two improvement techniques as follows:

- We observe that if a phrase contains numbers, it is often a sensitive phrase. It is because in most cases numbers represent sensitive information, such as telephone numbers, dates of birth, amounts of money, etc. As a result, we decided to consider all phrases containing numbers as sensitive phrases. In this way, we save time by not querying the Google for these phrases. In other words, prior to submitting a phrase to Google, we check whether numbers are contained in the phrase. If there are, the phrase is considered as sensitive eliminating the need to perform a Google search. By automatically considering phrases containing number as sensitive phrase, there is a potential of getting false positives. However, we note that this false positive rate causes by this case is very small and acceptable given that our method outperforms the traditional cyclical hashing method
- We propose that if we have two adjacent phrases that are overlapped by at least half of the length (this happens when $O < \frac{1}{2}C$ or in other words the length of the offset is less than half of the length of the candidate string), and one of these phrases is a popular phrase with a high score, we can also skip the Google search for the other. The reason is because with the significant overlapping between these two phrases, with high probability, the other phrase will not contain sensitive information.

Note that while these two improvement techniques help to further improve the processing speed, they may introduce false positives in some cases. Nevertheless, this very low false positive rate is tolerable when compared against the total accuracy.

V. Experimental Study

To evaluate the efficiency and effectiveness of IRILD, we implemented it in Python 2.5 and conducted experiments with the Enron Email Dataset [18], where we chose 50 emails from the "Inbox" of 10 employees as confidential documents (5 emails for each employee), and used 200 random emails in total from their "Sent_Items" to test the accuracy of the method. The emails used as confidential documents contained information both about company's internal procedures, such as scheduling employees' program, payment status, meetings, etc, and personal information such as telephone numbers, email addresses. We considered textual information in the Enron's website (an instance of January 2001 [19]) as public information. For comparison purpose, we compared IRILD to the popular approach that employed only cyclical hashing without the removal of public and common phrases in fingerprint generation.

Table I
Experimental settings

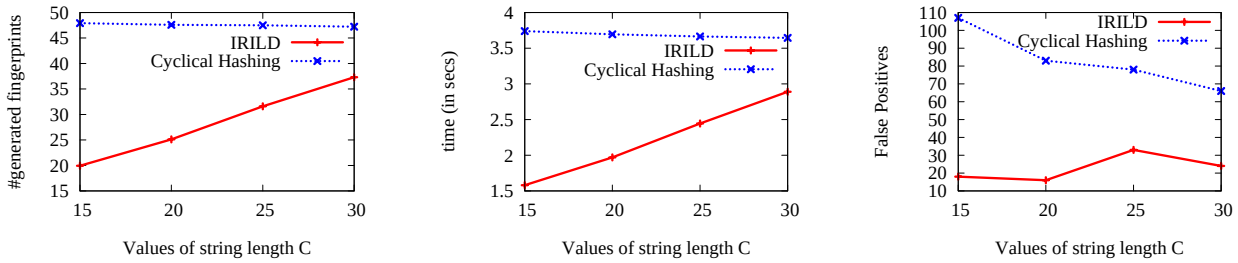
Parameter	Domain values	Default value
C	15 - 30	20
O	5 - 20	10
K	3 - 6	4

In the experiments, we set the default value of C to 20 because in our opinion the value of C should be equal to the average length of the queries made in Google. Note that since the average query in Google consists of 4 words[6] and the average length of an English word is 5 characters[7], setting C equals to 20 seems quite reasonable. We set the default value of O to 10 (which is half of C) because smaller values of O would create more fingerprints and thus the number of false positives would be increased. On the other hand, higher values of O would lead to the decrement of phrases that would be tested for information leaks, in other words there may be potential false negatives. Finally, we choose 4 as the default value of K because we believe that a phrase should be tagged as common if its occurrences at Google are beyond 10,000, in number. To summarize, the default and range of values of C, O, and K used in experiments are listed in Table I.

We evaluated the performance of IRILD and the popular approach in three aspects: (i) the cost of fingerprint generation in terms of processing time and the number of fingerprints, (ii) the cost of information leak detection in terms of processing time, and (iii) the accuracy of the results in terms of false positives and false negatives. Note that to compute the accuracy of IRILD and cyclical hashing, we manually looked at testing documents to extract all possible leaked cases (i.e., information that is copied from confidential documents).

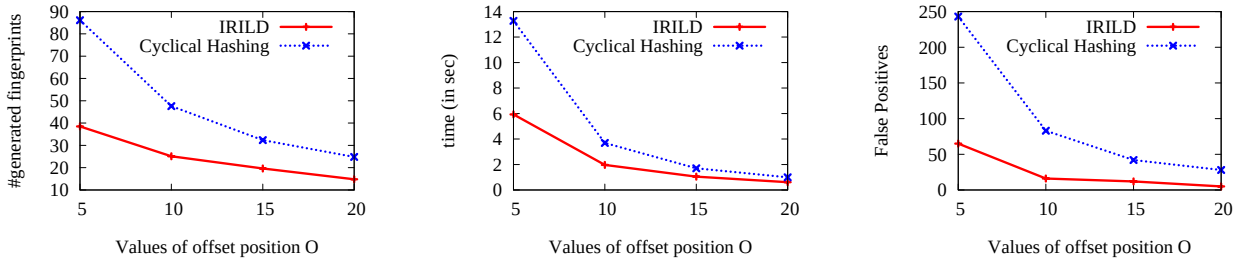
[6]http://www.beussery.com/blog/index.php/2008/02/google-average-number-of-words-per-query-have-increased/

[7]http://blogamundo.net/lab/wordlengths/



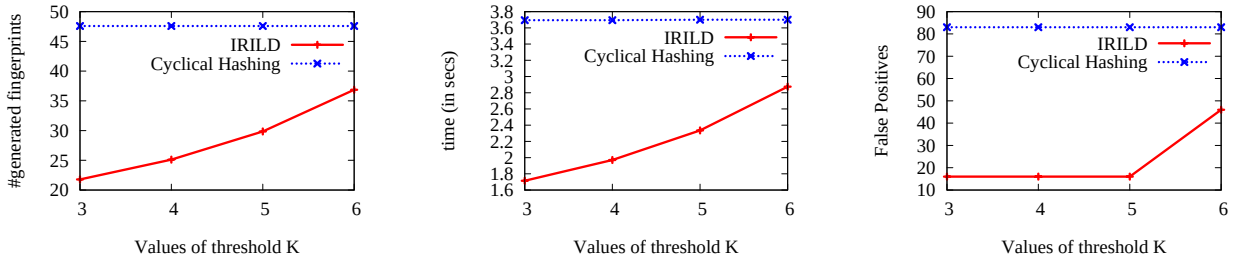
(a) on the number of fingerprints (b) on the leak detection time (c) on the accuracy

Figure 3. Effect of varying the string length C, keeping the offset position O = 10



(a) on the number of fingerprints (b) on the leak detection time (c) on the accuracy

Figure 4. Effect of varying the offset position O, keeping the string length C = 20



(a) on the number of fingerprints (b) on the leak detection time (c) on the accuracy

Figure 5. Effect of varying the threshold K, keeping the string length C = 20 and the offset position O = 10

A. *Cost of fingerprint generation*

We first evaluated the cost of fingerprint generation for confidential documents (emails in our case). As expected, IRILD incurred a processing time around 2 to 3 times longer than the basic approach, due to the extra time of submitting queries to Google to determine the popularity of candidate phrases. Even though IRILD takes time for indexing data, it happens only once. Later, the database of the fingerprints could be updated at intervals or when a new sensitive document is inserted to or an existing document is removed from the repository. Since the later step only involves one or few documents, it does not incur much processing cost. While this extra time can be reduced by employing PlanetLab [15] to submit queries concurrently, since fingerprint generation is done offline whenever a confidential document is submitted to the system, it does not affect the efficiency and effectiveness of IRILD.

The average number of fingerprints generated by IRILD and the basic approach when varying C and O are shown in Figure 3(a) and Figure 4(a) respectively. The results show that by removing public and common phrases from fingerprint generation, IRLID significantly reduced the number of generated fingerprints compared to the basic approach. In particular, as shown in Figure 3(a), the percentage difference in the number of generated fingerprints between cyclical hashing and IRILD varied from 21% when C = 30 to 58% when C = 15. On the other hand, as in Figure 4(a), IRILD generated 55% less fingerprints, at best case, that O = 5. Even at the worst case, when O

= 15, IRILD manages to generate 39% less fingerprints than cyclical hashing. It is interesting to observe that while the number of generated fingerprints in cyclical hashing does not depend on the value of C, in case of IRILD, the bigger the value of C is, the more the number of generated fingerprints is.

B. Cost of information leak detection

In this experiment, we measured the processing time required to detect information leak from testing documents. Figure 3(b) and Figure 4(b) show the processing time of information leak detection when varying C and O. As expected, since the number of fingerprints in IRILD was less than those in the basic approach, IRILD took a smaller number of comparisons, and hence it incurred a faster processing time. Actually, if we make a comparison between Figure 3(a) and Figure 3(b) as well as Figure 4(a) and Figure 4(b), the similarity between them show a strong correlation between the number of the generated fingerprints and the information leak detection time.

C. Accuracy

In order to check the accuracy of IRILD and cyclical hashing, fingerprints of confidential documents in the database are used to check against fingerprints of outgoing documents for information leak detection. So as discussed earlier, we evaluated the accuracy of IRILD by measuring false positives and false negatives, in terms of fingerprints. It is interesting to observe that we get the same number of false negatives in both IRILD and the basic approach. In particular, both methods have no false negatives when $O < 20$ and IRILD has only one false negative when O = 20, in contrast with cyclical hashing. That is because both approaches employ the same technique for indexing sensitive information and searching for information leaks. However, in terms of false positives, the result of IRILD is much better than that of the basic approach. As shown in Figure 3(c) and Figure 4(c), IRILD achieved a much better accuracy compared to the basic approach. As far as the true positives concerned, apart from the expected information leaks that came up, such as telephone numbers and some meeting dates, because they contained numbers, we also detected more information leaks. The first one contains the minutes of a meeting and the second one is about a contact with an outside partner.

It it important to note that in practice we would always set O to small values (e.g., $O < \frac{1}{2}C$) in order to avoid false negatives (i.e., all leaks should be detected). In this case, IRILD significantly outperforms the basic approach since according to this experiment and the previous two experiments in Sections V-A and V-B, the smaller the value of O is, the bigger the improvement IRILD has compared to the basic approach in terms of fingerprint generation cost, leak detection cost, and accuracy (or false positive cost).

D. Effect of varying the threshold K

So far we had K fixed at 4. In this experiment, we evaluated the effect of varying K from 3 to 6 on IRILD (note that the change of K does not affect the popular approach). The experimental results displayed in Figure 5 show that with the increasing of K, the number of generated fingerprints as well as the processing cost increased. It is because when K increased, we put a higher boundary for phrases to be considered common phrases, and hence less common phrases were identified and removed. The consequence of having less identified common phrases was an increase in the false positives of the method (happened when K = 6). Note that in the worst case if we set K to infinity so that all sentences are considered sensitive in IRILD, both IRILD and the basic approach will get the same number of false positives. Nevertheless, as discussed before, a reasonable value of K should not be high (e.g., 4 or at most 5).

VI. Conclusion

In this paper, we introduced IRILD, an information retrieval based solution to improve the performance of the traditional cyclical hashing approach for information leak detection. The core idea of IRILD is to *identify and remove* public phrases (found in public documents) and common phrases (identified by checking the number of results returned by Google when querying the phrases) from the fingerprinting process, since these types of phrases do not contain sensitive information. Furthermore, we conducted extensive experimental evaluation of our solution, and proved that it significantly outperformed the cyclical hashing approach. Specifically, IRILD achieved a much faster leak detection speed. As a result, IRILD can be utilized by systems with a large numbers of sensitive documents. Also, IRILD achieved much higher accuracy when compared with traditional cyclical hashing due to the removal of false positives related to public and common phrases.

References

[1] T. P. Institute, *2006 Annual Study: Cost of a Data Breach*, 2006, white Paper.

[2] S. Institute, *Understanding and Selecting a Data Loss Prevention Solution*, 2009, white Paper.

[3] P. T. Labs, *Information Leak Prevention Accuracy and Security Tests*, 2006, white Paper.

[4] J. Jung, A. Sheth, B. Greenstein, D. Wetherall, G. Maganis, and T. Kohno, "Privacy oracle: a system for finding application leaks with black box differential testing," in *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2008, pp. 279–288.

[5] R. Capizzi, A. Longo, V. N. Venkatakrishnan, and A. P. Sistla, "Preventing information leaks through shadow executions," in *ACSAC '08: Proceedings of the 2008 Annual Computer Security Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 322–331.

[6] N. Boufaden, W. Elazmeh, Y. Ma, S. Matwin, N. El-Kadri, and N. Japkowicz, "Peep – an information extraction base approach for privacy protection in email," in *Proceedings of CEAS*, 2005.

[7] V. R. Carvalho and W. W. Cohen, "Preventing information leaks in email," in *SDM*, 2007.

[8] C. Kalyan and K. Chandrasekaran, "Information leak detection in financial e-mails using mail pattern analysis under partial information," in *Proceedings of WSEAS*, 2007.

[9] P. Zilberman, A. Shabtai, and L. Rokach, "Analyzing group communication for preventing data leakage via email," in *Proceedings of COLLSEC*, 2010.

[10] B. Parno, J. M. McCune, D. Wendlandt, D. G. Andersen, and A. Perrig, "Clamp: Practical prevention of large-scale data leaks," in *SP '09: Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 154–169.

[11] R. Chow, P. Golle, and J. Staddon, "Detecting privacy leaks using corpus-based association rules," in *KDD*, 2008, pp. 893–901.

[12] B. D. Davison, D. G. Deschenes, and D. B. Lewanda, "Finding relevant website queries," in *In Proc. of WWW*, 2003.

[13] P. Nakov and M. Hearst, "Using the web as an implicit training set: application to structural ambiguity resolution," in *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Morristown, NJ, USA: Association for Computational Linguistics, 2005, pp. 835–842.

[14] J. Staddon, P. Golle, and B. Zimny, "Web-based inference detection," in *SS'07: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–16.

[15] *Planetlab: An open platform for developing, deploying, and accessing planetary-scale services*, www.planet-lab.org.

[16] *Blackbox Search*, http://www.blackboxsearch.com/.

[17] *Scroogle*, http://www.scroogle.org/.

[18] *Enron Email Dataset*, http://www.cs.cmu.edu/~enron/.

[19] *Internet Archive*, http://www.archive.org/.