A Novel Forensics Analysis Method for Evidence Extraction from Unallocated Space

Zhenxing Lei, Theodora Dule, and Xiaodong Lin

University of Ontario Institute of Technology, Oshawa, Ontario, Canada
{Zhenxing.Lei,Theodora.Dule,Xiaodong.Lin}@uoit.ca

Abstract. Computer forensics has become a vital tool in providing evidence in investigations of computer misuse, attacks against computer systems and more traditional crimes like money laundering and fraud where digital devices are involved. Investigators frequently perform preliminary analysis at the crime scene on these suspect devices to determine the existence of target files like child pornography. Hence, it is crucial to design a tool which is portable and which can perform efficient preliminary analysis. In this paper, we adopt the space efficient data structure of fingerprint hash table for storing the massive forensic data from law enforcement databases in a flash drive and utilize hash trees for fast searches. Then, we apply group testing to identify the fragmentation points of fragmented files and the starting cluster of the next fragment based on statistics on the gap between the fragments.

Keywords: Computer Forensics, Fingerprint Hash Table, Bloom Filter, Fragmentation, Fragmentation Point.

1 Introduction

Nowadays a variety of digital devices including computers and cell phones have become pervasive, bringing comfort and convenience to our daily lives. Consequently, unlawful activities such as fraud, child pornography, etc., are facilitated by these devices. Computer forensics has become a vital tool in providing evidence in cases where digital devices are involved [1].

In a recent scandal involving Richard Lahey, a former Bishop of the Catholic Church from Nova Scotia, Canada, the evidence of child pornography was discovered on his personal laptop by members of the Canada Border Agency during a routine border crossing check. Preliminary analysis of the laptop was first performed on-site and revealed images of concern which necessitated seizure of the laptop for more comprehensive analysis later. The results of the comprehensive analysis confirmed the presence of child pornography images and formal criminal charges were brought against Lahey as a result.

Law enforcement agencies around the world collect and store large databases of inappropriate images like child pornography to assist in the arrests of perpetrators that possess the images, as well as to gather clues about the whereabouts of the victimized children and the identity of their abusers. In determining whether a suspect's computer contains inappropriate images, a forensic investigator compares the files

X. Lai et al. (Eds.): E-Forensics 2010, LNICST 56, pp. 53–65, 2011.

from the suspect's device with these databases of known inappropriate materials. These comparisons are time consuming due to the large volume of the source material and so a methodology for preliminary screening is essential to eliminate devices that are of no forensic interest. Also, it is crucial that tools used for preliminary screening are portable and can be carried by forensic investigators from one crime scene to another easily to facilitate efficient forensic inspections. Some tools are available today which have these capabilities. One such tool created by Microsoft in 2008 is called Computer Online Forensic Evidence Extractor (COFEE) [2]. COFEE is loaded on a USB flash drive, and performs automatic forensic analysis of storage devices at crime scenes by comparing hash values of target files on the suspect device calculated on site with hash values of source files compiled from the law enforcement which we call alert database and stored on the USB flash drive. COFEE was created through a partnership with law enforcement and is available free of charge to law enforcement agencies around the world. As a result it is increasing prevalent in crime scenes requiring preliminary forensic analysis.

Unfortunately, COFEE becomes ineffective in cases where forensic data has been permanently deleted on the suspect's device, e.g., by emptying the recycle bin. This is a common occurrence in crime scenes where the suspect has had some prior warning of the arrival of law enforcement and attempts to hide evidence by deleting incriminating files. Fortunately, although deleted files are no longer accessible by the file system, their data clusters may be wholly or partially untouched and are recoverable. File carving is an area of research in digital forensics that focuses on recovering such files. Intuitively, one way to enhance COFEE to also analyze these deleted files is to first utilize a file carver to recover all deleted files and then runs COFEE against them. This solution is constrained by the lengthy recovery speed of existing file caring tools especially when recovering files that are fragmented into two or more pieces, which is a challenge that existing forensic tools face. Hence, the recovery timeframe may not be suitable for the fast preliminary screening for which COFEE was designed. Another option is to enhance COFEE to perform direct analysis on all the data clusters on disk for both deleted and existing files. However this option is again hampered by the difficulty in parsing files fragmented into two or more pieces.

Nevertheless, we can simply extract those unallocated space and leave those allocated space checked by COFEE. Then, similar to COFEE, we calculate the hash value for the data clusters of unallocated space. In order to cope with this design, each file in the alert database must be stored as multiple hash values instead of one in COFEE. As a result, the required storage space will be a very challenging issue. Suppose the alert database contains 10 million images which we would like to compare with files on the devices at the crime scene and suppose also that the source image files are 1MB in size on average. Assuming that the cluster size is 4KB on the suspect device, we can estimate the size of the USB device for storing all 10 million images from the alert databases. We assume that the result of a secure hash algorithm used is128-bit length, we would require 38.15GB storage capacity for all 10 million images. A 256-bit hash algorithm would require 76.29GB storage and a 512-bit hash algorithm such as SHA-512 would require 152.59GB (see Table 1). The larger the alert database, the larger storage space is needed for a USB drive such that 20 million images would require twice the storage previous calculated.

Table 1. The required storage space for different methods of storing alert database

Storage Method		10 million photos
Without using cryptographic hash		10^3GB
Hash Algorithms	SHA-256	76.29GB
	SHA-512	152.59GB

Motivated by aforementioned observations in terms of the size of the storage medium and the requirement for analysis of deleted files, we propose an efficient evidence extracting method which supplements COFEE. The contributions of this paper are twofold. First, we propose efficient data structures based on hash trees and Fingerprint Hash Table (FHT) to achieve both better storage efficiency and faster lookups. The FHT is a space-efficient data structure that is used to test the existence of a given element from a known set. Also, the hash tree indexing structure ensures that the lookups are fast and efficient. Second, we apply group testing technique based on statistics about the size of gaps between two fragments of a file [3] for effectively searching the unallocated space of the suspect device to extract fragmented files that were permanently deleted.

The rest of this paper is organized as follows: in Section 2 we briefly introduce some preliminaries and background knowledge. In Section 3 we present our proposal in detail and in Section 4 we discuss false positive rates and how we handle some special cases like unbalanced hash trees and slack space. In Section 5, we analyze the time complexity and storage efficiency of the proposed scheme. Finally, we draw our conclusions and directions for future work.

2 Preliminaries

In this section we will briefly introduce bloom filters and fingerprint hash table, which serve as important background of the proposed forensics analysis method for unallocated space. Then, we discuss file fragmentation issue and file deletion in file systems.

2.1 Bloom filter and Fingerprint Hash Table

A bloom filter is a hash based space efficient data structure used for querying a large set of items to determine whether a given item is a member of the set. When we query an item in the bloom filter, false negative matches are not possible but false positives occur with a pre-determined acceptable false positive rate. A bloom filter is developed by inserting a given set of items $E = \{e_1, \ldots, e_n\}$ into a bit array of m bits $B=(b_1, b_2 \ldots b_m)$ which is initially set to 0. K independent hash functions $(H_1, H_2 \ldots H_k)$ are applied to each item in the set to produce k hash values $(V_1, V_2 \ldots V_k)$ and all corresponding bits in the bit array are set to 1 as illustrated in Figure 1.

The main properties of a bloom filter are as follows [4]: (1) the space for storing the Bloom filter is very small as well as the size of a bit array *B*; (2) the time to query whether an element is in the Bloom filter is constant and is not affected by the number of items in the set; (3) false negatives are impossible, and (4) false positives are possible, but the rate can be controlled. As one space-efficient data structure for representing a set of elements, bloom filter has been widely used in web cache sharing [5, 6], package routing [7], and so on.

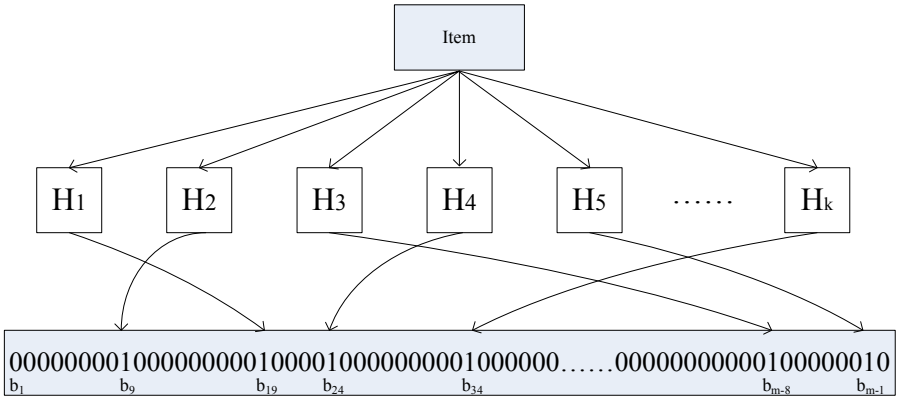

Fig. 1. m-bit standard Bloom filter

An alternative construction of Bloom filter is fingerprint hash table show as follows [8]:

$$\begin{cases} P(x): E \rightarrow \{1, 2, \ldots, n\} & (1) \\ F(x): E \rightarrow 1^{l} & (2) \end{cases}$$

Where P(x) is a perfect hash function [8] which maps each element $e \in E$ to an element at the unique location in an array of size n, F(x) is a hash function which calculates a fingerprint with $l=[\log 1/\varepsilon]$ bits of a given element $e \in E$, ε is the probability of a false positive, 1^{l} denotes a bit stream with a length l. For example, given the desired false positive probability of $\varepsilon=2^{-10}$, only 10 bits are needed to represent each element. In this case, the required storage space for the scenario in Table 1 is 2.98GB, which takes much less space compared to traditional cryptographic hash methods.

2.2 File System

2.2.1 File Fragmentation

When a file is newly created in an operating system, the file system attempts to store the file contiguously in a series of sequential clusters large enough to hold the entire file in order to improve the performance of file retrieval and other operations later on. Most files are stored in this manner but some conditions like low disk space cause

files to become fragmented over time and split over two or more sequential blocks of clusters. Garfinkel's corpus investigation in 2008 of over 449 hard disks collected over an 8 year period from different regions around the world provided the first published findings about fragmentation statistics in real-world datasets. According to his findings, fragmentation rates were not evenly distributed amongst file systems and hard drives and roughly half of all the drives in the corpus contained only contiguous files. Only 6% of all the recoverable files were fragmented at all with bifragmented files accounting for about 50% of fragmented files and files fragmented into three and as many as one thousand fragments accounted for the remaining 50% [3].

2.2.2 File Deletion

When a file is permanently deleted (e.g. by emptying the recycle bin), the file system no longer provides any means for recovering the file and marks the clusters previously assigned to the deleted file as unallocated and available for reuse. Although the file appears to have been erased, its data is still largely intact until it is overwritten by another file. For example, in the FAT file system each file and directory is allocated a data structure called a directory (DIR) entry that contains the file name, size, starting cluster address and other metadata. If a file is large enough to require multiple clusters, only the file system has the information to link one cluster to another in the right order to form a cluster chain. When the file is deleted, the operating system only updates the DIR entry and does not erase the actual contents of the data clusters [10]. It is therefore possible to recover important files during an investigation by analyzing the unallocated space of the device. Recovering fragmented files that have been permanently deleted is a challenge which existing forensic tools face.

3 Proposed Scheme

In this section we will first introduce our proposed data structure based on FHTs and hash trees for efficiently storing the alert database and fast lookup in the database. Then we will present an effective forensics analysis method for unallocated space even in the presence of file fragmentation.

3.1 Proposed Data Structure

3.1.1 Constructing Alert Database

In order to insert a file into alert database, we first divide the file size by 4096 bytes (cluster size) to create separate data items $\{e_1, e_2, e_3 \ldots e_n\}$ that are fed into P(x) so that we can map each element $e_i \in E$, $1 \leq i \leq n$, to a unique location in an array of size n. Later on, we store the fingerprint $l=[\log 1/\varepsilon]$ bits which is the F(x) value of a given element in each unique location. The process is repeated for the rest of the data items of each file; finally each file takes n*l bits in the alert database. In this manner, we store all the files into alert database.

3.1.2 Hash Tree Indexing

In order to get rapid random lookups and efficient access of records from the alert database, we construct a Merkle tree based on all cluster fingerprints of the files processed by the FHT and index each fingerprint as a single unit. In the Merkle tree,

data records are stored only in leaf nodes but internal nodes are empty. Indexing the cluster fingerprints is easily achieved in the alert database using existing indexing algorithms, for example binary searching. The hash tree can be computed online while the indexing should be completed offline when we store the file into the alert database.

Figure 2 shows an example of an alert database with *m* files divided into 8 clusters each. Each file in the database has a hash tree and all the cluster fingerprints are indexed. It is worth noting that in a file hash tree, the value of the internal nodes and file roots can be computed online quickly due to the fact that the hash value can be calculated very fast.

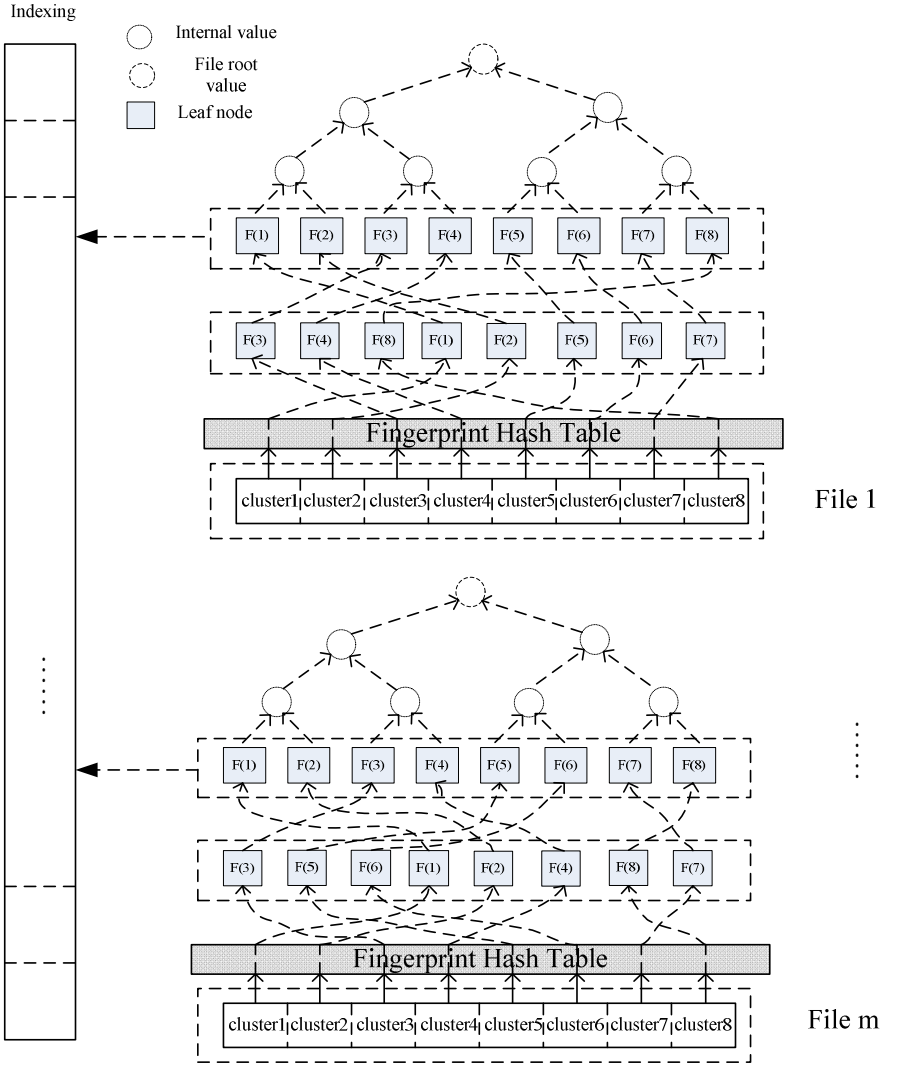

Fig. 2. Hash Tree Indexing

3.2 Group Testing Query Based on the Storage Characteristics

Group testing was first introduced by Dorfman [10] in World War II to provide efficient testing of millions of blood samples from US Army recruits being screened for venereal diseases. Dorfman realized that it was inefficient to test each individual blood sample and proposed to pool a set of blood samples together prior to running the screening test. If the test comes back negative, then all the samples that make up the pool are cleared of the presence of the venereal disease. If the test comes back positive however, additional tests can be performed on the individual blood samples until the infected source samples are identified. Group testing is an efficient method for separating out desired elements from a massive set using a limited number of tests. We adopt the use of group testing for efficiently identifying the fragmentation point of a known target file.

From Garfinkel's corpus investigation, there appears to be a trend in the relationship between the file size and the gap between the fragments that make up the file. Let us examine JPEG files from the corpus as an example. 16% of recoverable JPEG files were fragmented. With bifragmented JPEG files, the gap between the fragments were 8, 16, 24, 32, 56, 64, 240, 256 and 1272 sectors with corresponding file sizes of 4096, 8192, 12288,16384, 28672, 32768, 122880, 131072, and 651264 bytes as illustrated in Figure 3. Using this information, we can build search parameters for the first sector of the next fragment based on the size of the file which we know from the source database.

In limited case, the file is fragmented into two and more than two fragmentations. We suppose a realistic fragmentation scenario in which fragments are not randomly distributed but have multiple clusters sequentially stored. Under these characteristics, we can quickly find out the fragmentation point and the starting cluster of the next fragmentation.

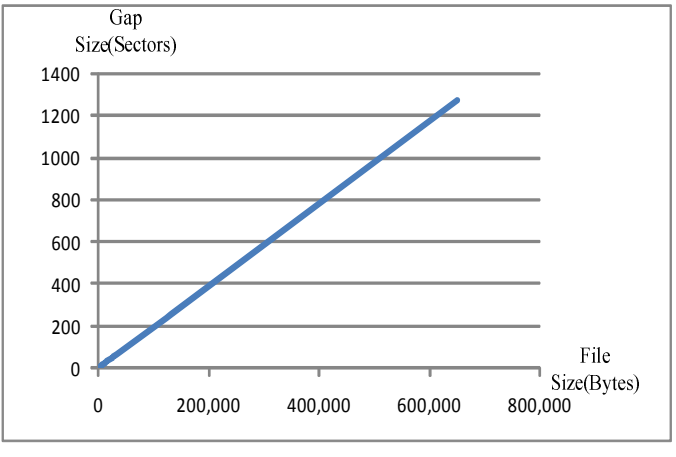

Fig. 3. The relation between the gap and the file size

3.3 Description of Algorithm

In the rest of this section, we discuss our proposed forensic analysis method with the assumption that the deleted file is still wholly intact and that no slack space exists on

the last cluster, which is considered the basic algorithm of our proposed scheme. Discussions on cases involving partially overwritten files and slack space trimming are presented in Section 4.

During forensic analysis when any cluster of a file is found in the unallocated space of the suspect's machine, we compute its fingerprint and search the alert database containing indexed cluster fingerprints for a match. If no match is found it means that the cluster is not part of the investigation and can be safely ignored. Recall that the use of FHTs to calculate the fingerprint guarantees that false negatives are not possible. If a match is found in the alert database then we can proceed to further testing to determine if the result is a false positive or a true match. We begin by checking if the target cluster is part of a contiguous file by pooling together a group of clusters corresponding to the known file size and then computing the root value of the hash tree in both the alert database and the target machine. If the root values match, then it means that a complete file of forensic interest has been found on the suspect's machine. If the root values do not match, then either the file is fragmented or the result is a false positive. For non-contiguous files, our next set of tests search for the fragmentation point of the file and as well the first cluster of the next fragment.

Finding the fragmentation point of a fragment is achieved in a similar manner as finding contiguous files with the use of root hash values. Rather than computing a root value using all the clusters that make up the file however, we begin with a pool of d clusters and calculate its partial root value and then compare it with the partial root value from the alert database. If a match is found, we continue adding clusters d at a time to the previous pool until there a negative result is returned which indicates that the fragmentation point is somewhere in the last d clusters processed. The last d clusters processed can then be either divided into two groups (with a size of d/2) and tested, or processed one cluster at a time and tested at each stage until the last cluster for that fragment, i.e., fragmentation point, is found.

In order to find the starting cluster of the next fragment, we apply statistics about gap distribution introduced in the previous section to select a narrow range of clusters to begin searching and perform simple binary comparisons using the target cluster fingerprint from the alert database. Binary comparisons are very fast and as such we can ignore the time taken for searching for the next fragment when calculating the

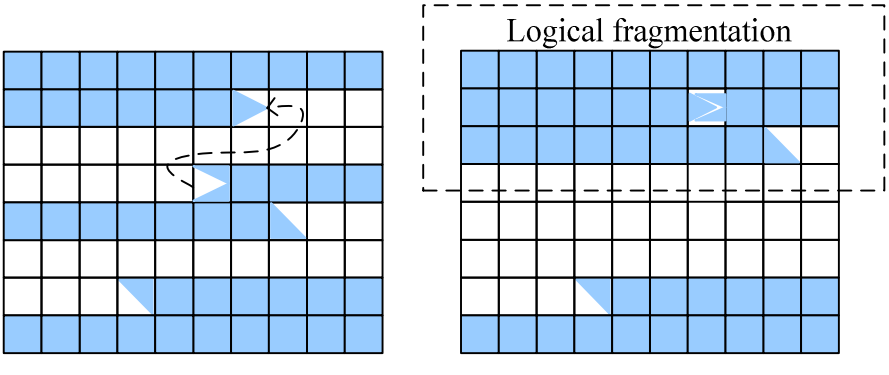

Fig. 4. Logical fragmentation for files of several fragments

time complexity. If the starting cluster of the next fragment cannot be successfully identified based on the gap distribution, brute-force cluster search is conducted on the suspect's device until a successful match occurs. Afterwards, the first two fragments are logically combined together by removing the clusters which separate them as shown in Figure 4 to form a single logical/virtual fragment. Verification of a match can be performed at this point using the aforementioned method for contiguous files. If the test returns a negative result, then we can deduce that the file is further fragmented. Otherwise, we successfully identify a file of interest.

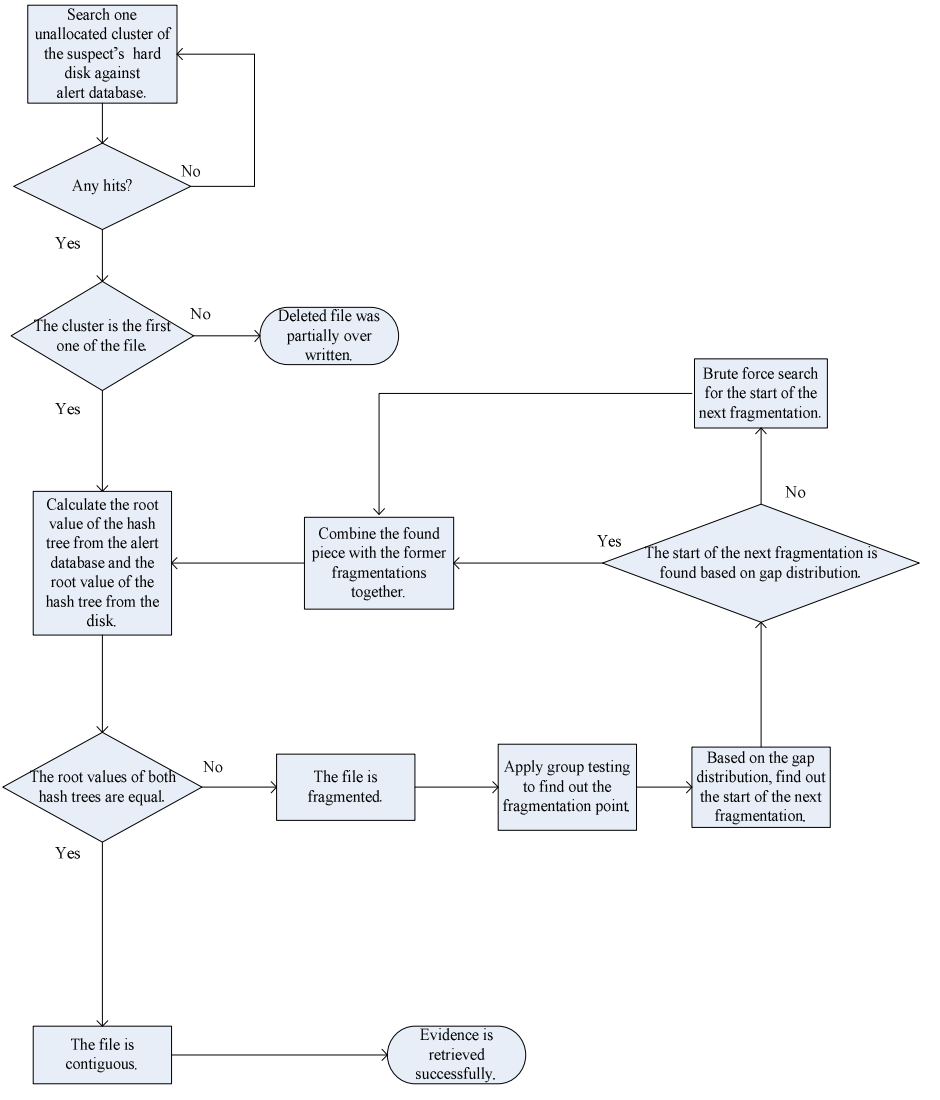

Fig. 5. The basic efficient unallocated space evidence extracting algorithm

Forensic analysis of contiguous files using this method has a time complexity of O (log (N)) while bifragmented files has a time complexity of O (log(N) + log(d)), where N=m*n, m is the total number of files in alert database, n is the number of clusters which each file in alert database contains. For simplicity, we consider the situation where the files in alert database have the same size. In the worst case where the second fragment of a bifragmented file is no longer available on the suspect's device (see Section 4 for additional discussion), every cluster on the device would be exhaustively searched before such conclusion could be reached. The time complexity in this case would be O(log(N) + log(d)+M), where M is the number of unallocated clusters on the suspect's harddisk.

For the small percentage (or 3%) of files that are fragmented into three or more pieces, once we logically combine detected fragments as a single fragment as illustrated in Figure 4, the fragmentation point of the logical fragment and the location of the starting cluster for the third fragment can be determined using statistics about the gap between fragments and binary comparisons as with bifragmented files. The rest of the fragmentation detection algorithm can follow the same pattern as bifragmenetd files until the complete file is detected. Figure 5 illustrates the efficient unallocated space evidence extracting algorithm discussed in this section.

4 Discussions

In this section we will discuss the effect of false positives from the FHT, handling unbalanced hash trees caused by an odd number of clusters in a file, and some special cases to be considered in the proposed algorithm.

4.1 False Positive in Alert Database

Bloom filter and it variants have a possibility of producing false positives where a cluster fingerprint from the alert database matches with a cluster fingerprint from the suspect's device that is actually part of an unrelated file. However, it could be an excellent space saving solution if the probability of an error is controlled. In fingerprint hash table, the probability of false positive is related to the size of the fingerprint representing an item. If the false positive probability is ε, the required size of the fingerprint is l=[log1/ ε] bits. For example, given the desired false positive probability of $\varepsilon=2^{-10}$, only 10 bits are needed to represent each element. Hence, The false positive ε' is shown in the function (3) when *d* cluster fingerprints from the alert database match with *d* fingerprints from the suspect's device but actually not

$$\varepsilon' = \varepsilon^d, \text{ where } l=[\log 1/\varepsilon] \tag{3}$$

The false positive will decrease when d or l increases. Therefore, we can simply choose the right d and l to control the false positive in order to achieve a good balance between the size of the cluster fingerprint and the probability of a false positive.

4.2 Unbalanced Hash Tree

An unbalanced hash tree will occur in cases where the clusters that form a file do not add up to a power of 2. In these cases, we can promote the node up in the tree until a

sibling is found [11]. For example the file illustrated in Figure 6 is divided into 7 clusters and the corresponding fingerprints are F(1), F(2), … F(7), but the value F(7) of the seventh cluster does not have a sibling. Without being rehashed, we can promote F(7) up until it can be paired with value *K*. The values *K* and *G* are then concatenated and hashed to produce value *M*.

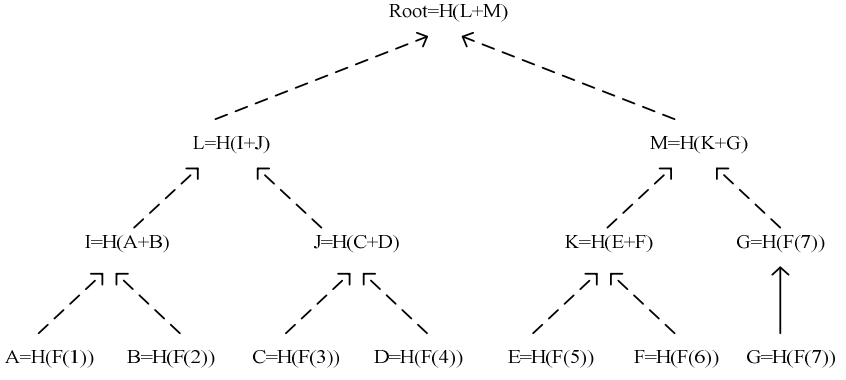

Fig. 6. An example of unbalanced hash tree

4.3 Slack Space Trimming

In a digital device clusters are equal-sized data units typically pre-set by the operating system. A file is spread over one or more clusters equal in size or larger than the size of the file being stored. This means that often there are unused bytes at the end of the last cluster which are not actually part of the file; this is called slack space. For example, on an operating system with 4 KB cluster size (4096bytes) and 512 byte sector, a 1236 byte file would require one cluster with first 1236 bytes containing file data and the remaining 2560 bytes are slack space as illustrated in Figure 7. The first two sectors of the cluster would be filled with file data and only 212 bytes of the third sector would be filled with data with the remaining 300 bytes and the entirety of clusters 4, 5, 6, 7 and 8 as slack space.

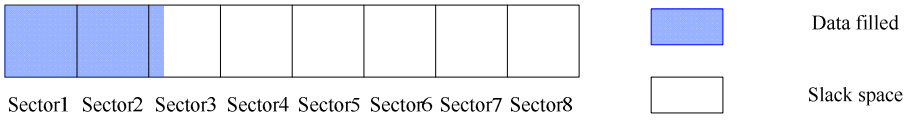

Fig. 7. Slack space in the cluster

Depending on the file system and operating system, slack space may be padding with zeros, may contain data from a previously deleted file or system memory. For files that are not a multiple of the cluster size, the slack space is the space after the file footer. Slack space would cause discrepancies in the calculated hash value of a file cluster when creating the cluster fingerprint. In this paper we are working on the assumption that the file size can be determined ahead of time from the information in

the law enforcement source database and as a result, slack space can be easily detected and trimmed prior to the calculation of the hash values.

4.4 Missing File Fragments

As discussed earlier when a file is deleted, the operating system marks the clusters belonging to the file as unallocated without actually erasing the data contained in the clusters. In some cases some clusters may have since been assigned to other files and overwritten with data. In these cases, part of the file may still be recoverable and decisions on how many recovered clusters of a file constitute evidence of the prior existence of the entire file is up to the law enforcement agencies. For example, a search warrant may indicate that thresholds above 40% are sufficient for seizure of the device for more comprehensive analysis at an offsite location.

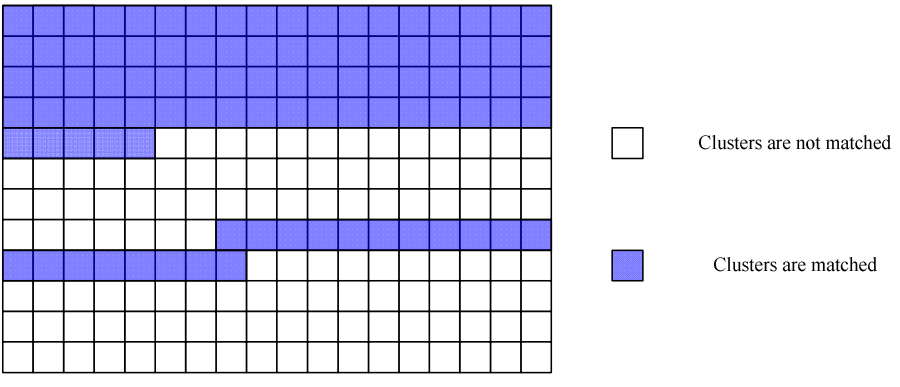

Fig. 8. 44.44% of one file are found, it can be seen as a warrant application evidence

Suppose the file in Figure 8 has four fragments and that the dark clusters (fragments 1 and 3) are still available on the suspect disk and the white clusters (fragments 2 and 4) have been overwritten with other information. Once the first fragment is detected using the techniques discussed in Section 3, detecting the second fragment will require the time consuming option of searching every single cluster when the targeted region sweep based on gap size statistics fails. After this search also fails to find the second fragment and we can conclusively say that the fragment is missing, we can either continue searching for the third fragment or prioritize these types of cases with missing fragments to the end after all other possible lucrative searches have been exhausted.

5 Complexity Analysis

Compared to the time complexity of the other query methods, such as classical hash tree traversal of O(2log(N)), where N=m*n, our proposed scheme is very promising as a result. Classical hash tree traversal for bifragmented files have a time complexity of O(2log(N)+2log(d/2)), and our scheme has only O(log(N)+log(d/2)). For file with multiple fragments the time complexity will be much more complicated as a result of utilizing sequential tests to query for the fragmented file cluster by cluster.

Nevertheless, very large fragments are typically seen only with very large files and the file information recovered from the first few during preliminary analysis may exceed the set threshold alleviating the need to continue exhaustive searching of the remaining fragments.

As we discussed in the section 4.1, when the false positive is 2^{-10}, the storage space for 10 million images each averaging 1MB is 2.98GB. It provides us a big advantage on choosing the storage device.

6 Conclusion and Future Work

In this paper we proposed a new approach to storing large amounts of data for easy portability in a space efficient data structure of FHT and used group testing and hash trees to efficiently query for the existence of files of interest and for detecting the fragmentation point of a file. The gap distribution statistics between the file fragments was applied to narrow down the region where searching for the next fragment begins. This approach helps us quickly query for relevant files from the suspect's device during preliminary analysis at the crime scene. After successful detection of target file using preliminary forensic tools that are fast and efficient, a warrant for further time consuming comprehensive analysis can be granted.

References

1. An introduction to Computer Forensics, http://www.dns.co.uk
2. Computer Online Forensic Evidence Extractor (COFEE), http://www.microsoft.com/industry/government/solutions/cofee/default.aspx
3. Garfinkel, S.L.: Carving contiguous and fragmented files with fast object validation. Digital Investigation 4, 2–12 (2007)
4. Antognini, C.: Bloom Filters, http://antognini.ch/papers/BloomFilters20080620.pdf
5. Fan, L., Cao, P., Almeida, J., Broder, A.: Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. In: ACM SIGCOMM 1998, Vancouver, Canada (1998)
6. Squid Web Cache, http://www.squid-cache.org/
7. Broder, A., Mitzenmacher, M.: Network Applications of Bloom Filters: A Survey, http://www.eecs.harvard.edu/~michaelm/NEWWORK/postscripts/BloomFilterSurvey.pdf
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. The MIT Press, Cambridge (2001)
9. Hua, N., Zhao, H., Lin, B., Xu, J.: Rank-Indexed Hashing: A Compact Construction of Bloom Filters and Variants. In: IEEE Conference on Network Protocols (ICNP), pp. 73–82 (2008)
10. Carrier, B.: File System Forensic Analysis. Addison Wesley Professional, Reading (2005)
11. Hong, Y.-W., Scaglione, A.: Generalized group testing for retrieving distributed information. In: IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Philadelphia, PA (2005)
12. Chapweske, J., Mohr, G.: Tree Hash EXchange format (THEX), http://zgp.org/pipermail/p2p-hackers/2002-June/000621.html