# Forensic Analysis of Unallocated Space

by

Zhenxing Lei

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Applied Science (MASc)

in

Electrical and Computer Engineering

Faculty of Engineering and Applied Science

University of Ontario Institute of Technology (UOIT)

June, 2011

## ABSTRACT

Computer forensics has become an important technology 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 suspects' devices to determine the existence of any inappropriate materials such as child pornography on them and conduct further analysis after the seizure of computers to glean leads or valuable evidence. Hence, it is crucial to design a tool which is portable and can perform efficient instant analysis. Many tools have been developed for this purpose, such as Computer Online Forensic Evidence Extractor (COFEE), but unfortunately, they become ineffective in cases where forensic data has been removed. In this thesis, we design a portable forensic tool which can be used to compliment COFEE for preliminary screening to analyze unallocated disk space by adopting a space efficient data structure of fingerprint hash tables for storing the massive forensic data from law enforcement databases in a flash drive and utilizing hash tree indexing for fast searching. We also apply group testing to identify the fragmentation point of the file and locate the starting cluster of each fragment based on statistics on the gap between the fragments. Furthermore, in order to retrieve evidence and clues from unallocated space by recovering deleted files, a file structure based carving algorithm for Windows registry hive files is presented based on their internal structure and unique patterns of storage.



**Keywords:** Computer Forensics; Fingerprint Hash Table; Bloom Filter; Fragmentation; Fragmentation Point; Registry Hive Files; Hive Bin; Key Cell.

## ACKNOWLEDDEMENTS

Looking back the days at UOIT, I have received tremendous help from lots of people. Otherwise, this dissertation would not have been completed.

I would like to convey my sincere gratitude to my supervisor Dr. Xiaodong Lin. It has been a great pleasure and fortune to have the opportunity to work with him. I would like to express my deep appreciation toward not only the financial support he offered, but also the advice on academic studies and life. I would also like to extend my appreciation to the examining committee members of my Master's thesis, Prof. Julie Thorpe, Prof. Mikael Eklund, Prof. Min Dong, and Prof. Ying Zhu, for the time and efforts they have given.

Moreover I would like to thank my understanding and supportive parents who provide me unconditional love during my graduate studies in Canada. Hopefully, this token of thankfulness is part of the payoff of their perseverance and hard-working.

Last but not least, thanks to all my friends here in Canada. Accompanied by all of you, I have the chance to be inspired and enjoy what Canada has offered me.

**TABLE OF CONTENTS**

## LIST OF FIGURES

## LIST OF TABLES

## ABBREVATIONS

| | |
|---|---|
| DNA | Deoxyribonucleic Acid |
| COFEE | Computer Online Forensic Evidence Extractor |
| USB | Universal Serial Bus |
| GB | Gigabyte |
| MB | Megabyte |
| SHA | Secure Hash Algorithm |
| HTML | HyperText Markup Language |
| XML | Extensible Markup Language |
| URL | Uniform Resource Locator |
| OS | Operating System |
| PC | Personal Computer |
| FHT | Fingerprint Hash Table |
| FAT | File Allocation Table |
| DIR | Directory Entry |
| PDF | Portable Document Format |
| RAR | Roshal Archive |
| CRC | Cyclic Redundancy Check |
| PUP | Parallel Unique Path |
| SHT | Sequential Hypothesis Testing |
| JPEG | Joint Photographic Experts Group |
| IE | Internet Explorer |
| FTK | Forensic Toolkit |

# Chapter 1 Introduction

## 1.1 Background

The internet and digital devices like portable media players, smart phones and digital cameras are now being used by many different people for a variety of reasons. Unfortunately, criminals are also designing their own uses to assist them in committing their crimes.

Law enforcement now has a great demand for investigators with a technology background to join the computer forensics field. This branch of forensic science is responsible for searching through and recovering evidence from digital sources. The evidence is not limited to files or emails found on a computer, but can be from any digital device that stores data. This field has become a vital tool in the fight against crime in cases where little or no physical evidence like DNA or fingerprints exists [1]. Unlike physical evidence, many crooks are aware of the digital trail they left behind and attempt to delete whatever evidence they can. For example, when the Enron scandal [2] surfaced in October 2001, top executives deleted thousands of e-mails and digital documents in an effort to cover up their fraud. Fortunately, when files on a computer hard drive are deleted the data isn't actually

erased. The space it occupies simply becomes available for use by other files which is also known as unallocated space. The less data added to the system after a file has been deleted greatly increases the possibility that the data can be recovered, even after long periods of time. But, as activities on the system increases, the likelihood that data can be fully recovered diminishes. This is because eventually these unallocated spaces will be re-assigned to other files.

Forensic analysis of unallocated disk space has attracted the interest of many forensic investigators as it has played an important role in the computer forensics field bringing convictions to so many important criminal cases. In fact, one of the decade's most fascinating criminal trials against corporate giant Enron was successful largely due to the digital evidence in the form of over 200,000 emails and office documents recovered from computers at their offices.

### 1.2 Motivations and Problem Statement

Law enforcement agencies around the world collect and store large databases of digital evidence, also known as alert databases. Child pornography, for example, is collected and saved to assist in the arrest and prosecution of the pedophiles that possess them. It's also used 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 from the suspect's device with these databases of known inappropriate materials. However, a simple bit-by-bit file comparison is very time-consuming due to the enormous size of today's storage devices. So a methodology for preliminary screening is essential to eliminate devices that are of no forensic interest. Also, as not every law enforcement agency has a specialized computer forensics agent, it is crucial that tools used for preliminary screening facilitate efficient forensic inspections. Some tools are available today which have these capabilities. One such tool, Computer Online Forensic Evidence Extractor (COFEE) [3], was created in 2008 by Microsoft. This software was created through a joint partnership with law enforcement [4] and is available at no cost for their use. This program is loaded on a USB flash drive, shown in Figure 1.1 and brought to a crime scene by forensic investigators to perform forensic analysis of live computer systems.


Figure 1.1 Computer Online Forensic Evidence Extractor (COFEE) [5]

COFEE works by comparing hash values of files on the target device with those compiled and stored on the flash drive. As a result, it is increasingly prevalent in crime scenes or even public spaces requiring preliminary forensic analysis, for example, border crossing

between countries. In a recent scandal involving Richard Lahey [6], 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.

Unfortunately, COFEE becomes ineffective in cases where forensic data has been removed from a device using simple methods like deleting viewed child pornography and emptying the Windows recycle bin. In these cases, COFEE cannot conduct a check on deleted files or unallocated spaces. This is a common occurrence in crime scenes where the suspect has had some prior warning of the arrival of law enforcement and attempted 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 (or unallocated spaces) is to first utilize a file carver to carve all deleted files and then run COFEE against them. This solution is constrained by the slow carving speed of existing file carving tools

especially when recovering files that are fragmented into two or more pieces, which is a challenge that existing forensic tools face. It is worth noting that recovering deleted files from a file system with residual file metadata is a simple task; many programs are available to the average home to do this. But, a savvy criminal can easily erase files in such a way that a tool which makes use of file system structure cannot recover. 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 disk spaces and leave those allocated spaces checked by COFEE. Then, similar to COFEE, we calculate the hash values for the data clusters of unallocated disk spaces. In order to cope with this design, each file in the database must be stored as multiple hash values instead of just one, like COFEE does. As a result, the required storage space needed to bring to the crime scene would limit the effectiveness of these tools. Suppose the alert database contains 10 million images which need to be compared with files found at a crime scene and suppose the source image files are 1MB in size on average, assuming that the cluster size is 4KB on the suspect device and the result of a secure hash algorithm used is 128-bit in length, we can estimate the size of the USB storage device to be 38.15GB. A 256-bit hash algorithm would require 76.29GB

of 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 previously calculated. Furthermore, several copies of alert database may be required due to the fact that different suspects' computers use different cluster sizes.

Table 1.1 Required storage space for different methods of storing alert databases

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

In this research, the focus is on designing a portable forensics tool which can be used to compliment COFEE for preliminary screening to analyze unallocated disk space.

In addition to images, a computer system may contain several other items of interests to a forensic investigator: Internet browsing history, instant messaging history (Windows Live Messenger, Skype, Facebook), audit and log files, run command history and the Windows registry. This sort of data is stored in files either using complicated proprietary formats, structured text documents like HTML and XML, or even simple delimited files. Proper

analysis of these areas could yield many leads or valuable evidence to both criminal and civil cases.

Some programs contain privacy settings which allow the user to remove any data which could later be used by someone else. The design of these features was to protect people using publicly available computers i.e. those in internet cafés, and hotel business centers. Nevertheless, the same tools can be used by criminals to hide their online activities by deleting browsing and searching history. On a Windows system, the Windows registry is the main storage area of the operating system (OS). It contains all the information the OS needs to track. This would include things like user accounts, file associations and way of tracking what the user has accessed. Previously typed URLs, previously accessed files, websites viewed, and run command history are all examples of the things stored in the Windows registry [7, 8]. Because of this, the registry is commonly manipulated or deleted by privacy tools or through manual means. Therefore, it is crucial to recover deleted Windows registry files during a digital investigation involving a Windows system if applicable after preliminary tests reveal suspicious activities where suspects have left. In the past, the recovery of a Windows registry heavily relies on file metadata. However, it becomes very challenging when Windows registry file metadata is missing, especially, because of the following reasons:

a) The internal structure of Windows Registry file is not well documented;

b) Windows Registry files could be fragmented into hundreds of pieces, which makes currently existing file carving methods ineffective.

A survey we conducted shows that the Windows Registry files of 52% of Windows systems we studied are fragmented. In other words, file fragmentation exists commonly on Windows Registry files. Additionally, as no PC has had the same set of files added, deleted, updated, or moved, the distribution of these fragments is completely random across the entire disk, even if a defragmentation program has been executed.

In essence, we are also investigating how to effectively carve deleted Windows registry files out of unallocated disk space based on the internal structure of the Windows registry files as well as their unique patterns of storage.

### 1.3 Objectives and Research Contributions

The objective of this research focuses on designing an efficient evidence extracting method from unallocated space, which supplements COFEE, to help investigators extract evidence from a Windows computer. In addition, this study also aims to develop a novel file carving algorithm for Windows registry files.

The contributions of researching an efficient evidence extracting method are twofold. First, we propose to use data structures based on hash tree index and Fingerprint Hash Table (FHT). The FHT is a well organized storage efficient data structure that can be applied to test the existence of a given element from a known set. The hash tree indexing structure ensures that the lookups are fast and efficient. Second, we apply group testing techniques based on statistics about the size of gaps between two fragments of a file [9] for effectively searching the unallocated space of the suspect's device to extract fragmented files that were permanently deleted.

The novel file carving algorithm for Windows registry files is based on a survey of 50 computer systems. The registry hive files on each computer were used to verify the structure of this data repository, which is designed by Microsoft but hasn't been released publicly. Additionally, we supplemented more newly discovered details to clarify the structure and the fragment characteristics based on the statistics from the survey to help us get a more accurate recovery result. Based on the survey result, we proposed a novel file carving algorithm for Windows registry files based on their internal structure and unique patterns of storage. It overcomes the challenges facing traditional file carving approaches caused by file fragmentation.

## 1.4 Organization of Thesis

The rest of the thesis is organized as follows. Chapter 2 describes related work. In Chapter 3, we introduce the novel forensics analysis method for evidence extraction from unallocated space. Chapter 4 provides the novel file carving algorithm for Windows registry files. Finally, Chapter 5 presents conclusions and directions for our future work.

# Chapter 2 Related Work

The increased use of digital devices in the developed world has led to the creation and rapid expansion of the Computer Forensics field of investigation within law enforcement. As more people, including criminals, use devices with digital storage capabilities, the discovery of such devices at crime scenes increases; as does the need to analyze these devices for evidence.

Computer forensics is the "analysis including a manual review of material on the media, reviewing the Windows registry for suspect information, discovering and cracking passwords, keyword searches for topics related to the crime, and extracting e-mail and pictures for review" [10]. This science provides evidence for investigations in all types of crimes. This ranges from individuals who use personal computers for illegal purposes like child pornography or computer hacking, to groups of people who use multiple devices to commit crimes like money laundering and fraud.

The disk analysis tools used by forensic investigators frequently examine the unallocated space of storage devices, often containing deleted files which are no longer referenced as active by the file system [11].

In point of fact, when a file is permanently deleted in the file system, the file system no longer provides any means for retrieving the file and marks the clusters previously assigned to the deleted file as unallocated hence available for reusing by other files. 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 or directory is allocated a data structure called a directory entry (DIR) 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 thus form a cluster chain. As shown in Figure 2.1, File.txt has a size of 16000 bytes. Suppose that the cluster size is 8 sectors (or 4096 bytes), File.txt occupies the clusters 36-39. We can see that the starting cluster address in the directory entry indicates the first cluster 36 of the file and the FAT structure has the cluster chain which connects all the remaining clusters of File.txt.

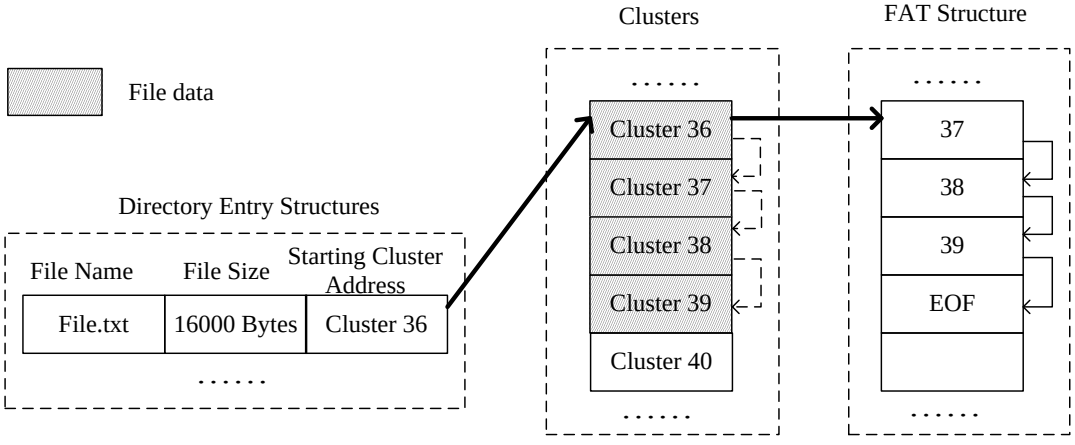


Figure 2.1 The file in allocated state on hard disk

When the file is deleted, the operating system only updates the DIR entry where the first byte of the directory entry (or the first character of the file name) is set to a special character (0xe5 in hex). At the same time, it changes the FAT entries for the rest clusters to “0”. But the operating system does not erase the actual contents of the data clusters. As shown in Figure 2.2, it shows its relevant remaining information in the system after File.txt (in the example of Figure 2.1) deleted from the system. The two cardinal changes are: in directory entry, “File.txt” updates to “_ile.txt” and the FAT cluster chain is wiped out.

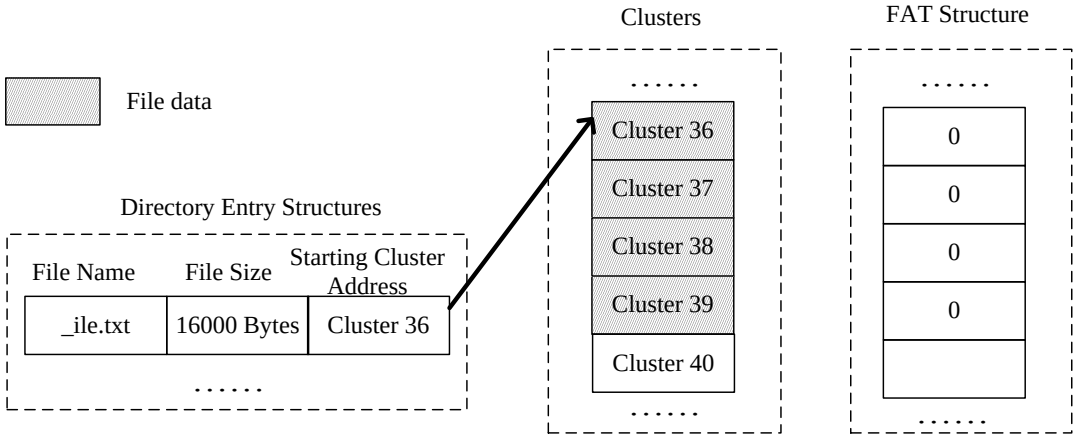


Figure 2.2 The file in unallocated state in hard disk

Obviously, it is very straightforward to recover deleted files by using the remaining file metadata information in a FAT file system given that most of files are stored in contiguous blocks. First, we simply change the first character of the file name back to its original one. Then, we put back the cluster chain in FAT since we know the starting cluster address from DIR entry as well the file is stored on consecutive disk blocks.

Currently, there are a number of both open source and commercially available tools which allow the recovery of deleted or hidden files from within the unallocated spaces of a storage device. Tools such as EnCase [12], WinHex [13], and Forensic Toolkit [14] use a basic search of the file system for any residual data left after a file is deleted. These tools can identify how big a file is and where on the device the beginning of the file is located. These tools will recover the first cluster plus however many additional clusters are needed to match the size of the file. There is a small difference between how these tools will decide which of the additional clusters are needed. WinHex v11.25, for example, is able to blindly recover all clusters regardless of their allocation status in the file system. Other tools like EnCase, can pull only unallocated data from within the specific cluster chain as listed in the FAT [15]. These tools can be used for the most basic types of file recovery only. These tools represent a very basic form of data recovery; once the files becomes fragmented or lose their reference from within the file system's meta-data, the software won't be able to recover the files properly.

Because of these limitations, a more complex software that doesn't rely on the file system's meta-data was created. Such "File Carving" software was designed to reconstruct files based solely on their contents. Similarly to the previously mentioned tools, when recovering non-fragmented files with a file carver, the process is very fast. Many files have unique header/footer structures and many tools like Foremost [16] and Scalpel [17] utilize

this to recover data. The Foremost software package was developed first and it works by first creating a configuration file which contains file header information. It then tries to match each of the headers with a corresponding footer. Unfortunately, this software will repeatedly search through data that has previously been matched, increasing the length of time a few searches can take. Richard et al. proposed Scalpel [18] to fix this performance issue by creating a high performance multiple file systems carver. Foremost spent much of its time reading and writing to the hard drive, often the slowest piece in a computer system. The improved file carver, Scalpel, first indexes all headers and footers, then looks for potential matches from within that index which is stored in memory; a much faster method than repeatedly searching the hard drive. Additionally, the software also contains improved memory-to-memory copy operations, as well as faster byte writing output. These improvements made Scalpel a much more efficient file carving tool. These two tools, however, make no effort to validate the recovered data, so false positive header/footer matches result in corrupt file recoveries.

Except for these well-known work based on header and footer recovering, many detailed work focusing on some specific file types have also been proposed, such as carving the RAR file [19], the PDF file [20]. Since RAR file is the most commonly archived file, the authors designed the carving algorithm based on the information and internal structure of the RAR. They applied mapping functions to locate the header and footer of an RAR file,

comparing the size of the file in the RAR file with the distance between the header and footer of the RAR file or the file size to determine whether the file is fragmented. After they applied enumeration to reassemble the two fragments which were extracted, they implemented the CRC of decompressed data stored in the file header to validate the integrity of RAR file which is a good reminder for us to do the file validation after a file is extracted.

All the tools previously discussed are great at the recovery of contiguously stored files, but issues arise when trying to recover files that have been split into multiple pieces and stored in different locations across on the disk. While the traditional set of meta-data based tools previously discussed become ineffective, file carving tools can still recover data. That being said there are challenges that must be overcome to recover these fragmented files.

When a file is fragmented into two pieces, one piece contains the file header and the other file footer. In [9], this was identified as a bi-fragmented file. A new approach was proposed to recover this particular type of file, called bi-fragment gap carving, which is based on a dedicated survey. The survey from more than 300 hard drives used on the second hand market shows 50% of recovered fragmented files are bi-fragmented files. Moreover, the survey shows that the gap between the first fragment and the second fragment is a relatively small number of disk sectors. Based on these observations, this process first identifies

where the file header and footer are located. The file header is considered the starting point of the first fragment and the file footer is considered the ending point of the second fragment. A gap, “G”, of sectors containing non relevant data must therefore exist between the two pieces of a bi-fragmented file. Once the program identifies the proper number of sectors to remove, the file can be properly recovered. An initial value of G is assigned and tested from either side of the gap. This value sequentially increases until the correct file sequence is found.

For example, as shown in Figure 2.3, a bi-fragmented file takes up three clusters and the initial value assigned to G is 2. Step 1 begins with removing the first two clusters right after the file header. The remaining clusters would be concatenated and tested. When the test result of step 1 is negative, the algorithm moves one cluster forward and repeats its concatenation and testing; this would be step 2. This process repeats until the algorithm reaches the file footer. At this point the value of G would be increased and the process would restart removing the appropriate amount of clusters after the file header. In the example, when G has a value of 3, the third step would result in a positive match, and a proper extract of the file can be made. After the file is recovered, several object validation designs were introduced to validate the carved file in an effort to eliminate false positives.

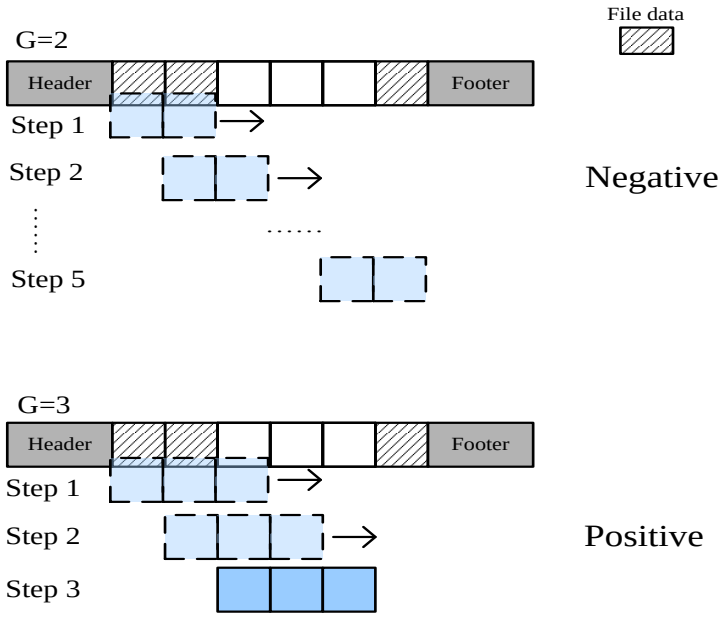


Figure 2.3 Using gaps to concatenate the file

Although this work proposes a promising carving algorithm, it assumes having the correct file header and footer. Moreover, all aforementioned file carvers become ineffective when dealing with files that have more than two fragments. The challenge in file carving turns out to be how to recover the file with more than two fragments.

In recent years, several works focused their attention on those files with more than two fragments. For example, Memon et al. [21] created a vertex disjoint graph which reassembled the fragments by ordering them according to the compared weights of the clusters from all the potential fragments, where the weight of two clusters indicates how likely they are physically adjacent. In order to reconstruct the file, they introduced the greedy heuristic model to help in choosing the best candidate. Then, they proposed eight

different algorithms for reassembling the fragments based on this greedy heuristic. The best of which, the greedy parallel unique path (PUP) algorithm, reconstructed the file without relying on the order of the images being reconstructed.

Based on Memon et al's previous work, in 2008, Pal et al. [22] presented detecting file fragmentation point using sequential hypothesis to detect the fragmentation point of a file by sequentially comparing adjacent pairs of blocks from the first block of the file until the fragmentation point is detected. As the solution in Garfinkel's paper [9] repeated, decoding is the method to identify the starting point of the next fragment of the file. In order to recover the file with multiple fragments, they refined the file carver by combining the modified parallel unique path algorithm (PUP) with sequential hypothesis testing (SHT) to retrieve multiple fragmented files.

In order to decrease the highly intensive method of looking for the starting block of the remaining fragments, Memon et al. proposed [23] an algorithm based on bit sequence matching to distinguish fragments generated by the same Huffman code tables. First, they clearly defined the JPEG file format. Then, based on the internal structure, they designed an algorithm which compares entropy-coded data of the fragment bit sequences generated by the same Huffman code tables to find the file fragments. This reduced the complexity of the computation and made it possible to recover files with many fragments.

Although the method of looking for the starting block of the next fragment is the same as Garfinkel's work [9], this bit sequence matching can work on fragmented files with missed fragments, something Garfinkel's method could not solve.

The above-mentioned work mainly focuses on how to carve out an image file stored in various places on the disk. There is additional work which has proposed file carving for PDF, RAR, IE and audit log files. Missing, or at least lacking, are papers discussing the Windows registry hive files. These are the operating system's main data repository. And not surprisingly, they can contain direct evidence or at least clues a forensic investigator can use. Available research related to a Windows registry is limited. In [24], the author gives a brief overview of the internal structure of the Windows registry. This is the first explanation of the registry based on both physical and logical data units.

Following that, Morgan published [25] which is a more complicated review of the Windows NT registry file format. Based on the information from this work, Morgan [26] explained how Windows deletes the registry data, then provided the algorithm to recover deleted cells; the basic unit the Windows registry uses to store its data. Jolanta Thomassen's thesis [27] described a process that uses the very low-level binary information of registry hive files to analyze the unallocated space for recovering remnants of information left in the registry hive files. Unfortunately, both of Morgan and

Thomassen's work directly focuses on extracting data units from the registry, which doesn't provide a complete picture of the registry files an investigator needs to gather their evidence from.

An alternative method would involve accurately extracting the hive files first, which would then allow a straightforward method to carve the required data from registry files. But, as previously mentioned, there is little work available which focuses on how to recover Windows registry files. Two issues stand in the way of understanding the process needed to properly recover these files:

a) Microsoft has yet to publish any related work which details the specifics of how the registry data structures are organized on the computer's disk.
b) File fragmentation of the registry files is extremely common even after defragmentation software has been run. In fact, registry hive files can be fragmented into as many as one hundred pieces.

All Windows system related files, including the registry files, are stored on disk in a specific range which is fairly easy to locate and then extract in a short time period. Contrary to Window's system files, application and user files can be stored randomly across the device's storage media. Locating and identifying these files can be a very lengthy process, and increasingly so, as the capacity of hard drives and other storage media continuously

doubles. At the time of this writing, home users have access to terabyte size storage devices and some enterprise businesses are backing up petabyte of data. Using the previously mentioned tools, a forensic investigator would simply not have enough time to analyze all unallocated space of the computer clusters making up the new forms of cloud computing.

A process that can first scan and classify the type of data stored in the unallocated space would help to overcome some of these time constraints. Once the data is classified, the investigator can choose which type of data is of interest and then focus on recovering only that data. For example, Figure 2.4 shows both allocated and unallocated clusters on a section of a computer disk. The unallocated clusters in this Figure have been classified showing that they contain data from various types of files including jpg, mp3 and pdf files. Once the information has been classified, it can then be grouped together and the data of interest can then be extracted for further analysis.

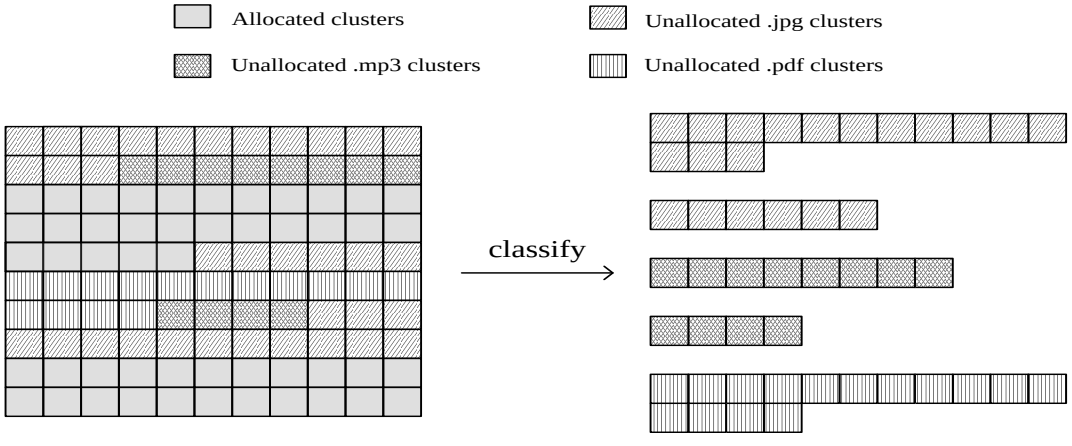


Figure 2.4 Unallocated clusters classification

Calhoun et al [28] and Veenman [29] published methods which are both based on this methodology. That is, one which tries to predict file types from within file embedded meta-data. These can be a big asset for an investigator trying to carve files from large storage media. More specifically, Calhoun et al [28] investigated two algorithms to determine the type of a file fragment. One uses linear discrimination and the other uses the longest common subsequences. The linear discrimination is a statistical classification method used to classify individuals into groups based on Kolmogorov complexity. The idea of the longest common subsequences is that cluster of data with the same file type, probably have similar longer substrings. Close to their design, Cor J. Veenman uses the statistical data of the clusters to predict the file types. Basically, Veenman uses the statistical content of the cluster to get the content features. Based on the content features, the cluster prediction problem becomes a classification problem. After sufficient clusters of a file have been extracted, a specialized model can be established to help locate the rest of the file's cluster based on analysis of the already recovered clusters.

These papers, theoretically, are fundamental to the practice. But there is a huge gap between these designs and the requirements of law enforcement. In reality, a forensic investigator may have millions of files pending analysis. If their files are deleted and still exist in unallocated space, it is inefficient to use the previously mentioned tools or designs to first recover the files, then compare each of them against their reference database.

Instead, a fast lookup system should be built which directly searches through unallocated space and compares its contents directly with the information stored within the police databases. The tool COFEE is developed for the investigator to do the preliminary analysis at the crime scene, but unfortunately, COFEE becomes ineffective in cases where forensic data has been permanently deleted on the suspect's device, i.e. emptying the recycle bin or if the operating system has removed as well. Because of these drawbacks of COFEE, we propose our design in the later section as a supplement of COFEE, to search the unallocated space of a storage device which no longer has a file system.

Based on the above description of the available tools, there is still a gap between what the forensic investigator needs and what is available. Most of these works are suitable for analyzing and carving the files within a lab environment, and many still focus on only bi-fragmented files. Even though many works consider the validation of a file after it has been recovered, seldom do papers discuss files recovered with missing fragments or how to deal with file fragments that have been damaged or overwritten.

In our research, we design a tool which is portable, but more importantly a tool that is fast enough to investigate digital storage media at the crime scene, particularly unallocated space on digital storage media. To meet these requirements, we introduce the efficient data structures based on hash trees and Fingerprint Hash Table (FHT) to achieve both better

storage efficiency and faster lookups. Moreover, we use group testing techniques along with statistics about the size of gaps between two fragments of a file for effectively searching the unallocated space in an effort to extract fragmented files. This tool will be able to discover multiple fragmented files and files not stored in a sequential order. False positives can be controlled in the design as well. Also, we design an algorithm for recovering the Windows registry file based on its internal structure and fragment characteristics, which are discovered using a survey of 50 computers with different versions of the Windows Operating system. Most importantly, in our designs, we considered how to deal with files with missing or overwritten sectors.

# Chapter 3 Forensic Analysis and Evidence Extraction of Unallocated Space

## 3.1 Preliminaries

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

### 3.1.1 Bloom Filter

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*=($b_1$, $b_2$ ... $b_m$), where each bit is initially set to 0. *K* independent hash functions ($H_1$, $H_2$ … $H_k$) that are applied to each item

in the set to produce *k* hash values ($V_1$, $V_2$ … $V_k$) and all corresponding bits in the bit array are set to 1 as illustrated in Figure 3.1.

The main properties of a bloom filter are as follows [30]: (1) the space for storing the Bloom filter is very small; (2) the time to query whether an element is in the Bloom filter is the same 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 [31, 32], package routing [33], and so on.

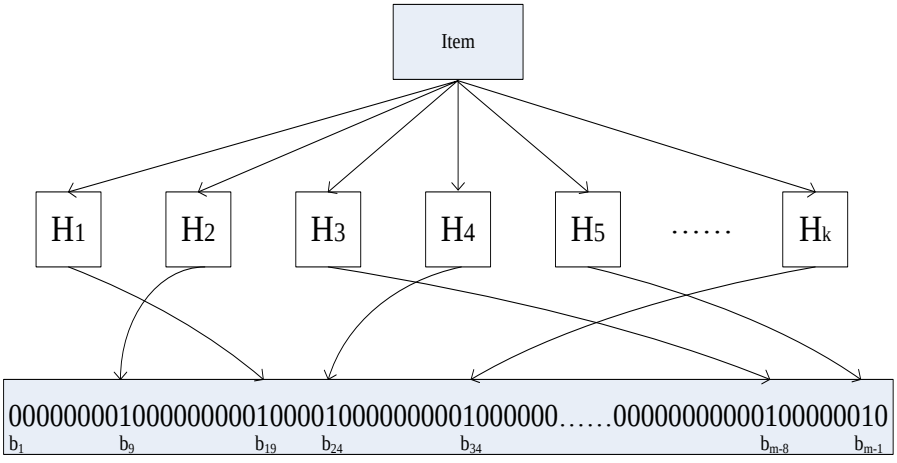


Figure 3.1 m-bit standard Bloom filter

An alternative construction of a Bloom filter is the fingerprint hash table shown as follows [34]:

$$\begin{cases} P(x): E \rightarrow \{1, 2, \ldots, n\} & (3.1) \\ F(x): E \rightarrow 1^{l} & (3.2) \end{cases}$$

Where P(x) is a perfect hash function 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 $e \in E$, $\varepsilon$ 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 1.1 is 2.98GB, which takes much less space compared to traditional cryptographic hash methods.

### 3.1.2 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 [9]. 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 bi-fragmented 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%.

### 3.1.3 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 [35]. It is therefore possible to recover important files during an investigation by analyzing the unallocated space of the device.

## 3.2 Proposed algorithm for evidence extraction from unallocated space

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

### 3.2.1 Constructing Alert Database

Law enforcement agencies around the world collect and store large sets of inappropriate images like child pornography, and can be used to build up a database, also known as alert database, 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.

Now, we will introduce how to build up an alert database in an efficient way. In order to insert a file into an alert database, we first divide the file size by 4096 bytes to create an array of 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 another array of size n. It is worth noting that file cluster size can be different for different file systems. For simplicity, we assume the cluster size is 4 KB. 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 the alert database.

### 3.2.2 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 fingerprint hash table 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 3.2 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. 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 calculations are very fast [36].

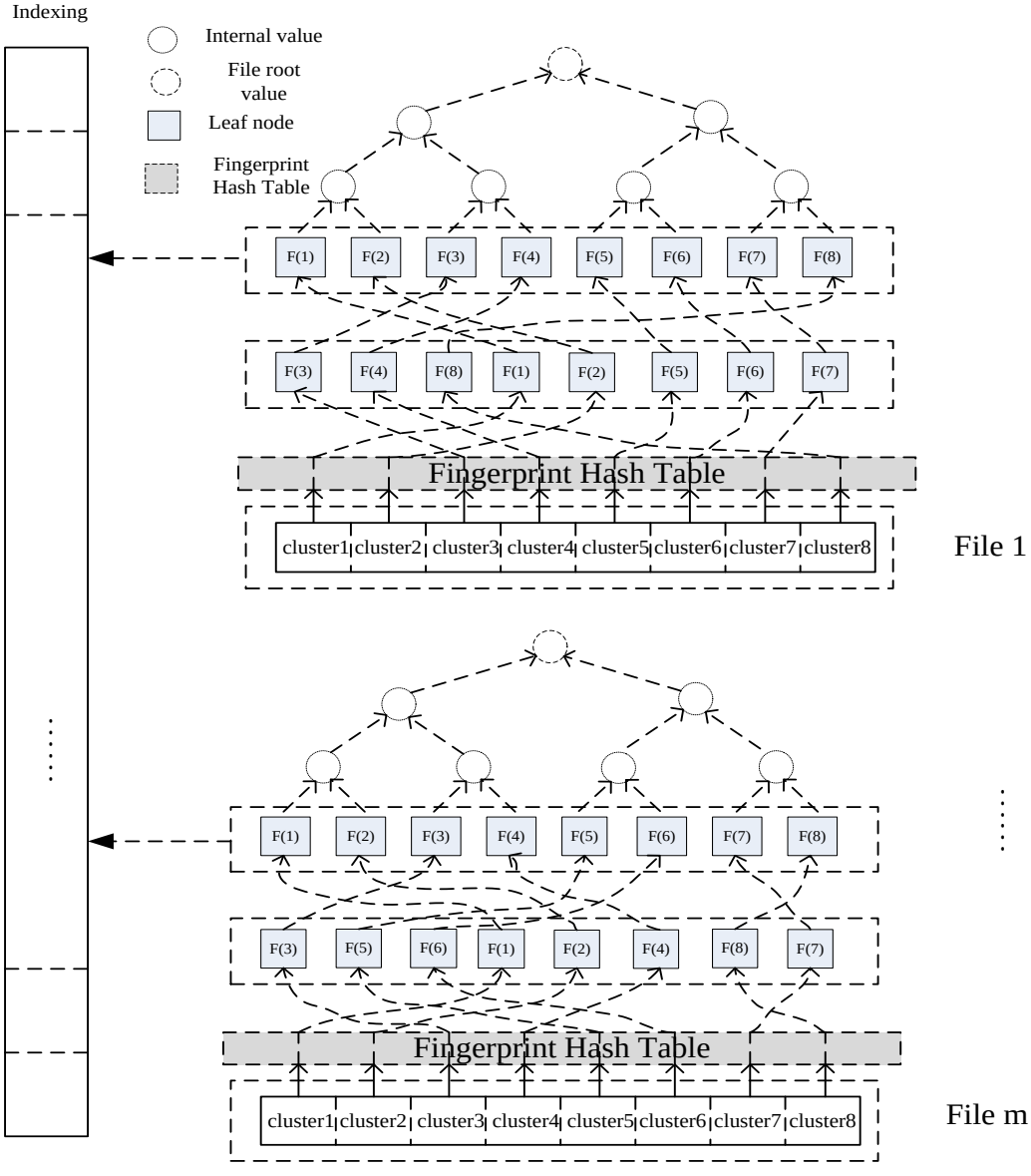


Figure 3.2 Hash Tree Indexing

### 3.2.3 Group Testing for Evidence Extraction

Group testing was first introduced by Dorfman [37] 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 which is stored on a suspect's computer and has been deleted.

From Garfinkel's corpus investigation [9], there appears to be a trend in the relationship between the file size and the gap between the fragments that make up the file. For example, as for JPEG files from the corpus investigated by Garfinkel, 16% of recoverable JPEG files were fragmented. With bi-fragmented 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 3.3.

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 or alert database.

In limited cases, the file is fragmented into two and more than two fragments. 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.

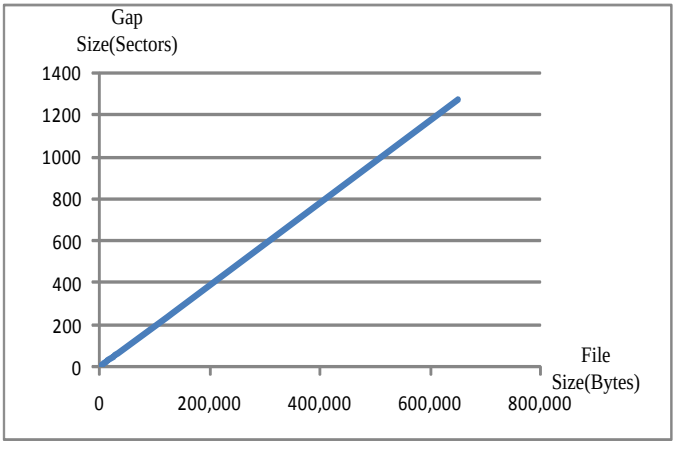


Figure 3.3 The relation between the gap and the file size

### 3.2.4 Description of Algorithm

In this sub-section, we illustrate our proposed forensic analytical method to effectively extract evidence from unallocated disk space 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. Slack space is the unused space in the last cluster for a file. Discussions on cases involving partially overwritten files and slack space trimming are presented in Section 3.3.

During forensic analysis when any cluster is extracted from the unallocated space of the suspect's machine, we compute its fingerprint and search the alert database containing indexed cluster fingerprints of known inappropriate files 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 fingerprint hash table 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 the file is fragmented. For non-contiguous files, our next set of tests search for the fragmentation point of the file and then the first cluster of the next fragment.

Finding the fragmentation point of a fragment is achieved by group testing procedures 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 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.

Next, we need to find the first cluster of the next fragment. 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 of the target cluster fingerprint with the one 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 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 identified are logically combined together by removing the clusters

which separate them as shown in Figure 3.4 to form a single 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, a complete file of forensic interest that has been fragmented into two pieces (or is bi-fragmented) has been found on the suspect's machine.

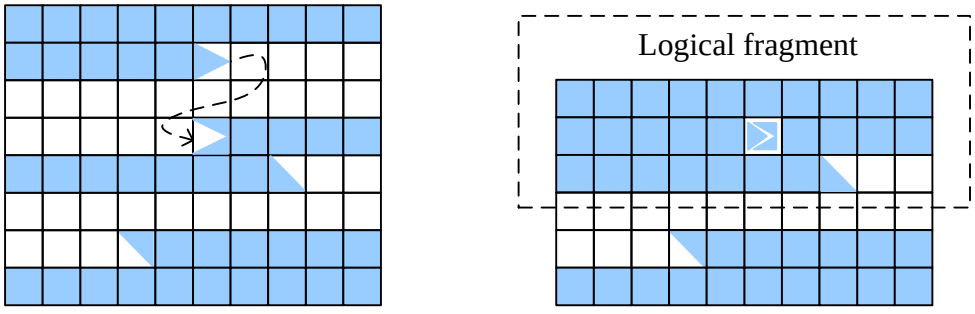


Figure 3.4 Logical fragment for files of several fragments

For a small percentage (or 3%) of files that are fragmented into three or more pieces, once we logically combine detected fragments as a single virtual fragment as illustrated in Figure 3.4, the fragmentation point detection of the logical fragment can be achieved by group testing procedures 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 bi-fragmented files. The rest of the fragmentation detection algorithm can follow the same pattern as bi-fragmented files until the complete file is detected. Figure 3.5 shows the main steps of the efficient unallocated space evidence extracting algorithm.

Step 1: Scan unallocated space on suspect's disk until a match is found;
Loop:
Step 2: Check if the file is contiguous or fragmented by comparing the root values of hash trees from the alert database and suspect's disk. If the file is contiguous, break;
Step 3: Find the fragmentation point by using group testing;
Step 4: Find the starting cluster of the next fragment based on the gap distribution. If not successful, a brute-force search is adopted;
Step 5: Combine two identified fragments logically by removing irrelevant data between these two fragments;
Loop ends
Step 6: Evidence is retrieved.

Figure 3.5 The main steps of the algorithm

Next, we will use an example to further illustrate how the proposed algorithm works. Suppose the cluster size used in the suspect's hard disk is 4KB. One suspicious image is stored on suspect's disk and the size of the file is 32K bytes. As shown in Figure 3.6, the image is fragmented into three pieces, being separately distributed on suspect's hard disk. This image was later deleted by the suspect. Clusters 1-4 are holding the image clusters C1 to C4 and clusters 7-9 are used to hold the image clusters C5 to C7 and cluster 12 is holding image cluster C8. For this situation, it will be very challenge for existing file carving tools to carve out this deleted image due to the fact that this file has been fragmented into more than two pieces. However, our design can easily find it out as follows.

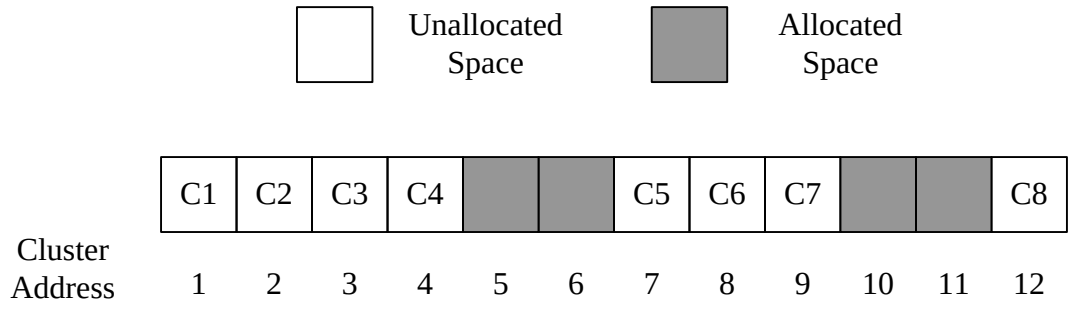


Figure 3.6 The suspect image on unallocated space

First, we scan the disk until we have a match, where C1 is found in unallocated spaces of suspect's disk. Then we move to Step 2, test if the file is contiguous or fragmented which is the beginning of the loop. According to the file size from the alert database, we can group 8 contiguous clusters together in Figure 3.7 starting from cluster 1, which contain clusters 1 to 8 on suspect's hard disk, and calculate the root hash value of these clusters. Afterwards, we compare it with the root hash value calculated from the alert database. The comparing result is false because there are two clusters not belonging to the file. Then we know the file is fragmented and we go to next step, find out the fragmentation point.

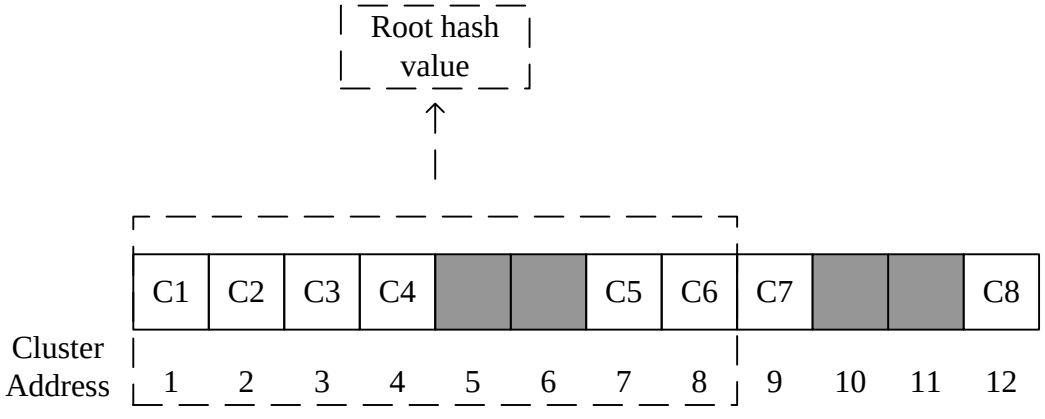


Figure 3.7 Root hash value of eight clusters

According to the binary splitting group testing, we group 4 clusters together physically from cluster 1 to cluster 4 and calculate the partial root hash value, and compare it with the partial root hash value from the alert database. Although the comparing result is true, but we cannot yet confirm whether cluster 4 is the fragmentation point. Then we do further analysis, and group these 4 plus half of the half which is 6 clusters, from cluster 1 to cluster 6 shown in Figure 3.8 (a). Obviously, the result is false. We know the cluster which doesn't belong to the image is one of the last two clusters, either cluster 5 or 6. In other words, we know that the fragmentation point is located at cluster 4 or 5. Then we use binary splitting group testing to test 5 clusters, which are the clusters 1 to 5 shown in Figure 3.8 (b). The result is also not true. It means that cluster 4 is the fragmentation point.

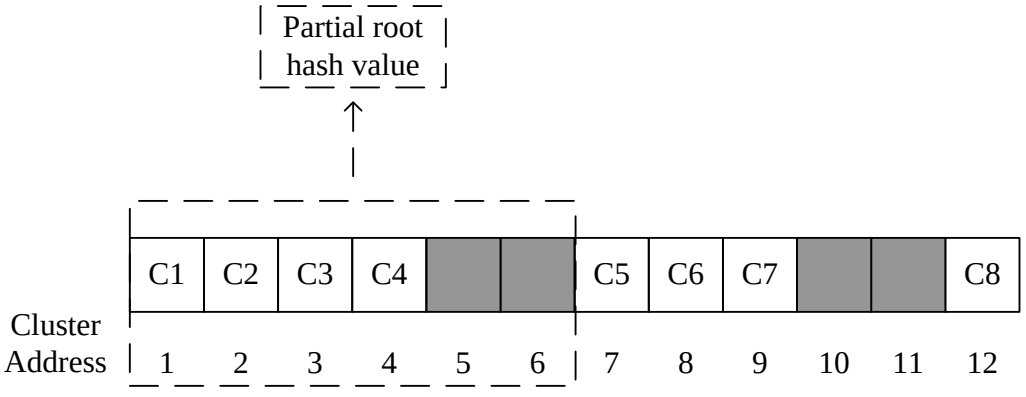


Figure 3.8 (a) Six clusters' partial hash tree

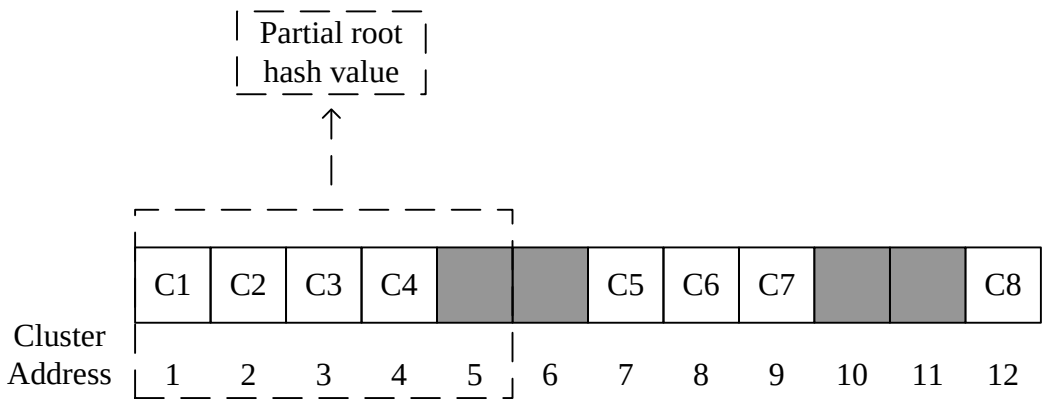


Figure 3.8 (b) Five clusters' partial hash tree

After we know cluster 4 is the fragmentation point, we need to find the starting cluster of the next fragment, which is cluster 7 that contains the content of C5 of the image. For simplicity, we assume that according to the gap distribution, we can locate this cluster very quickly. After we find the starting cluster of the second fragment, we logically combine these two fragments together and make it to be a virtual contiguous fragment shown in Figure 3.9. Later on, we go back to the beginning of the loop, check if the file is a complete file or further fragmented by comparing the root hash values from the alert database and the suspect's disk. Since the root hash values still do not match, we know the file is further fragmented.

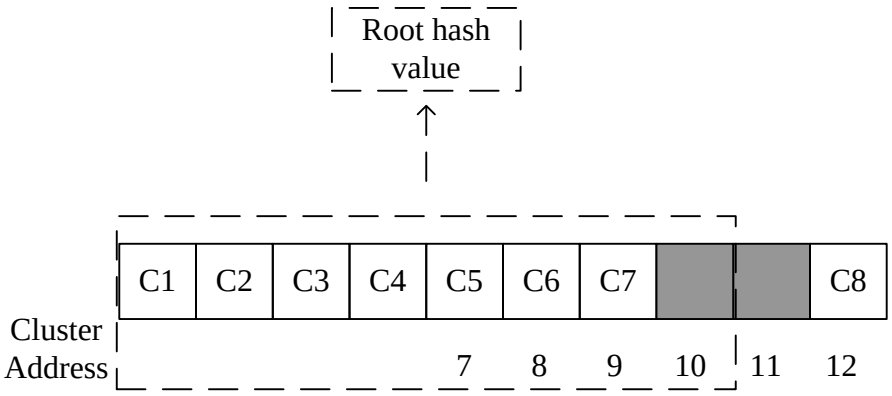


Figure 3.9 Eight clusters' hash tree after combining

Following the same way, we can locate the fragmentation point, i.e., cluster 9, and the starting cluster of the next fragment is cluster 12. At the end, the root hash value of the file matches the one from the alert database. Therefore, we can say we find the evidence, a known inappropriate image, on suspect's disk.

### 3.3 Discussions

In this section we will discuss the effect of false positives from the fingerprint hash table, handling unbalanced hash trees caused by an odd number of clusters in a file, and some special cases to consider in the proposed algorithm, such as missing file fragments, file slack space.

### 3.3.1 False Positive in Alert database

The Bloom filter and its 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 legitimate file. However, as an excellent space saving solution, the probability of an error can be 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 (or cluster). Hence, the false positive ε' is shown in the function (equation 3.3) when *d* cluster fingerprints from the alert database match with *d* fingerprints from the suspect's device but actually do not.

$$\varepsilon' = \varepsilon^{d}, \text{ where } l = [\log 1/\varepsilon] \qquad (3.3)$$

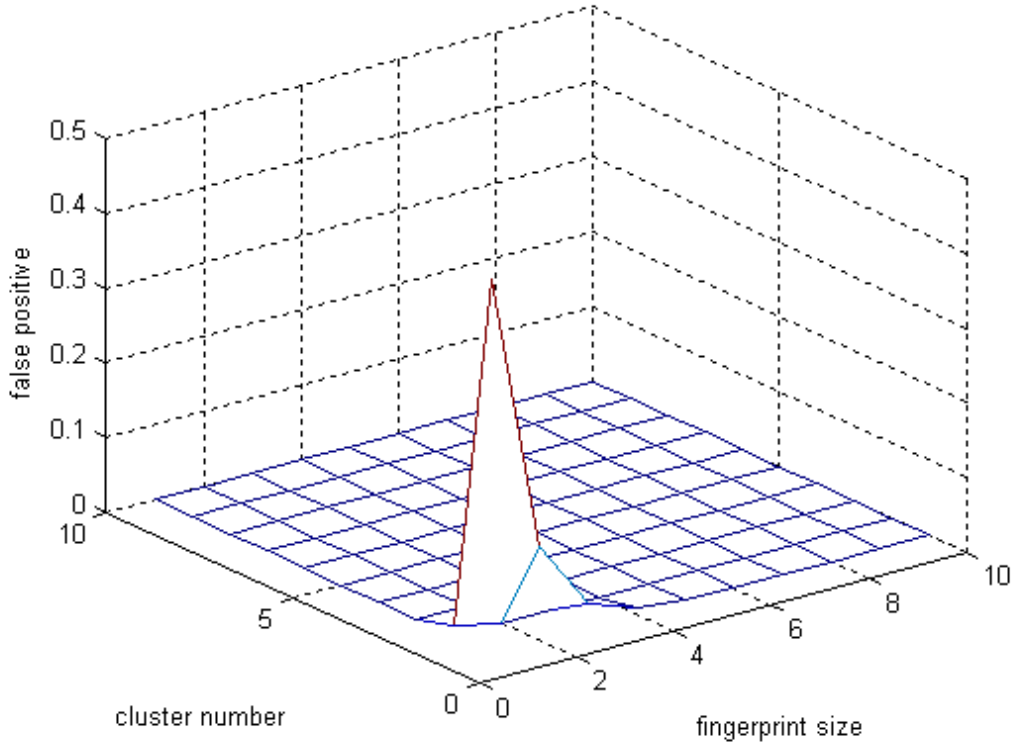


Figure 3.10 The relationship between the false positive and parameters

From Figure 3.10, we can see the false positive will decrease when d or $l$ increases. As $l$, the size of the cluster fingerprint, or d, the cluster number, increases, the false positive decreases, but the storage requirement for alert database 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 storage requirement and the probability of a false positive.

### 3.3.2 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 [38]. For example the file illustrated in Figure 3.11 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*.

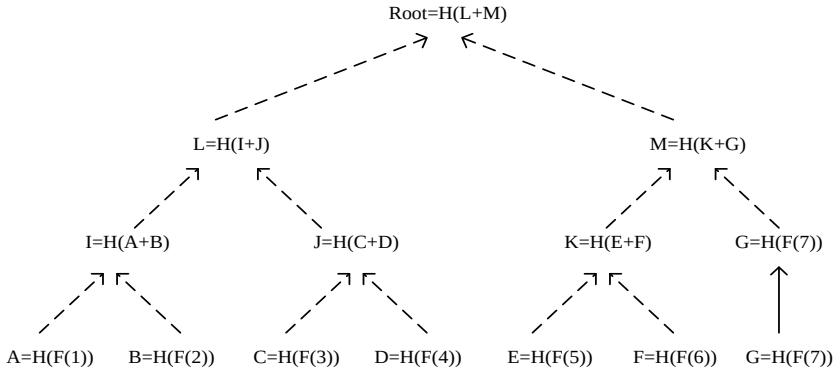


Figure 3.11 An example of unbalanced hash tree

### 3.3.3 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 3.12. 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 sectors 4, 5, 6, 7 and 8 as slack space.

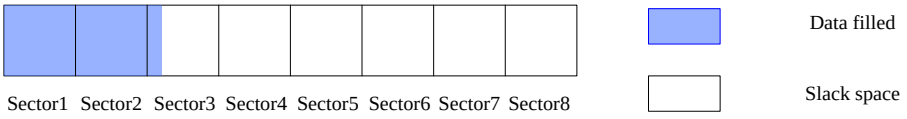


Figure 3.12 Slack space in the cluster

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 work, we are working on the assumption that the file size can be determined ahead of time from the information in the law enforcement source (alert) database and as a result, slack space can be easily detected and trimmed prior to the calculation of the hash values.

### 3.3.4 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.

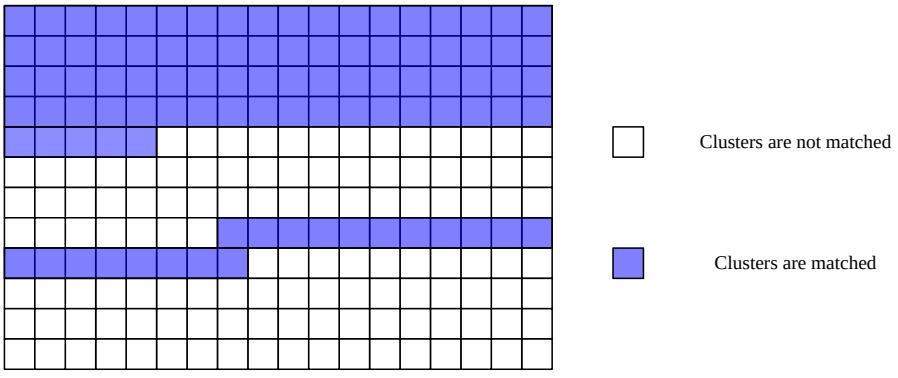


Figure 3.13 The potential evidence

Suppose the file in Figure 3.13 has four fragments and that the colored clusters (fragments 1 and 3) are still available on the suspect disk while 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 3.2, 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 cluster or prioritize these types of cases with missing fragments to the end after all other possible lucrative searches have been exhausted.

### 3.4 Complexity Analysis

Unlike the traditional way which used to identify invalid samples on large sample spaces with the minimal number of validation and verification tests, in our application, the objective of group testing is to efficiently identify the fragmentation point of a known target file which is stored on a suspect's computer and has been deleted.

Obviously, if a file is not fragmented and stored contiguously, the computational complexity of the proposed algorithm is O(1). In the following, we will discuss the complexity of the proposed algorithm when deleted files on suspects' hard disks are fragmented.

Even though the computational complexity of group testing for a general purpose has not been determined [39], several group testing algorithms have been studied in [40], including individual testing, binary splitting, and Li's s-stage algorithm. Table 3.1 gives the bound for the number of tests (in the worst case) to identify the fragmentation point using different

group testing algorithms. Specifically, n is the number of clusters which a file occupies and m is the number of segments which a file has.

Table 3.1 The number of required tests for different group testing algorithms

| Algorithm | Tests (m=2) | Tests (m>2) |
|---|---|---|
| Individual testing | n-1 | n-1 |
| Binary splitting | $\lceil \log(n) \rceil$ | $(m-1)\ \lceil \log(n) \rceil$ |
| Li's s-stage Algorithm | $\frac{e}{\log e} \log(n)$ | $(m-1)\frac{e}{\log e} \log\left(\frac{n}{m-1}\right)$ |

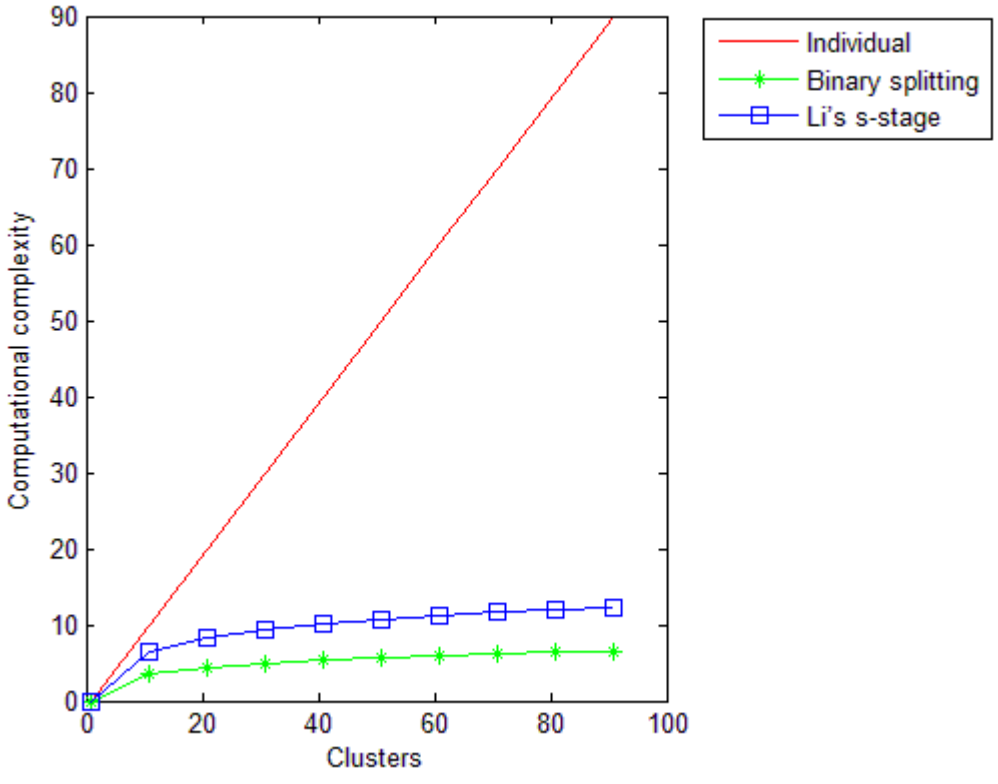


Figure 3.14 Computational complexity of different group testing algorithms

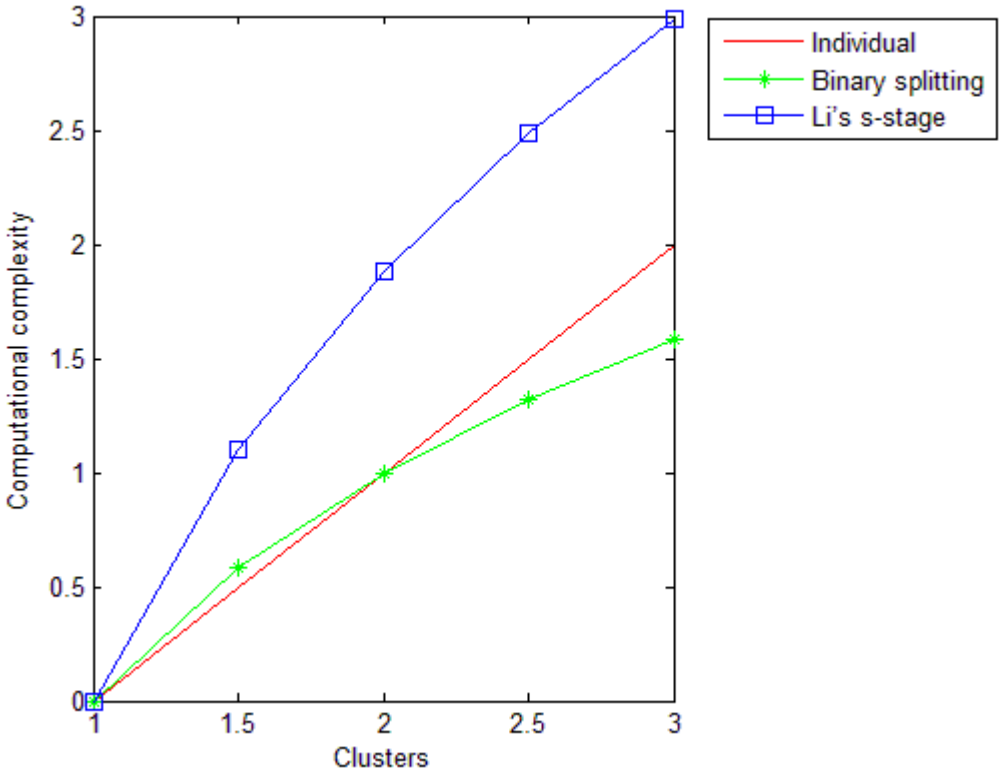


Figure 3.15 Enlarged part of Figure 3.14

Figure 3.14 shows the number of tests (hash value comparison) required as n changes. Figure 3.15 shows enlarged image when the cluster number of the file is between 1 and 3. From Figure 3.15 we can see, when the cluster number is less than 3, the individual testing is faster than the binary splitting. However, the binary splitting algorithm is the most efficient algorithm when files are bigger. In reality, an image file can contain up to several MB or the number of the clusters needed is larger. Hence the binary splitting algorithm is the best group testing method for our application scenario. In sum, for a file with m fragments, the computational complexity for searching the whole will be $O((m-1)\ \lceil \log(n) \rceil)$.

# Chapter 4 File Structure based Carving Algorithm for Windows Registry

## 4.1 Windows Registry

In this section, we first briefly introduce Windows Registry, including its hierarchical structure, Hive File, and hive bins (HBINs).

In Windows, the registry is made up by a set of discrete files called hives, also known as registry hive files or hive file [41]. Each hive contains a set of keys, which are organized in a hierarchical structure, as shown in Figure 4.1.

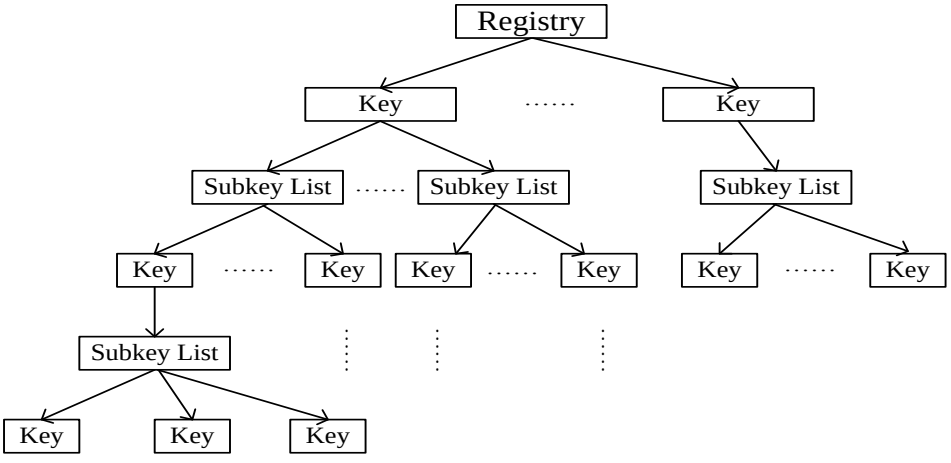


Figure 4.1 Hierarchical structure of registry

### 4.1.1 Hive File Organization

Each hive is divided into a set of allocation units (or blocks or clusters), where the first one is called base block with a signature “regf” in its header and the rest are organized into a set of hive bins (HBINs). Each HBIN can be made up by either one or several blocks and always start with a unique signature “hbin”. HBINs are linked together through length and offset fields defined in their headers. In addition, each HBIN implicitly indicates the starting point of the next HBIN through its size in its header. In Figure 4.2, the dotted line arrows in the logical layout show a HBIN links from itself to its next one. More importantly, from the first HBIN, 4 bytes in each HBIN header are used to clarify the offset to the first HBIN where the field contains all zeroes for the first HBIN, indicating this is the first HBIN. The solid line arrows in the logical layout in the Figure 4.2 show the reference points between each HBIN and the first HBIN. In addition to the HBIN header, it is made up by a group of cells, which are the basic containers used to store the registry data and linked together through its offset field. The HBIN layout in Figure 4.2 shows simple links from the first key cell to different cells.

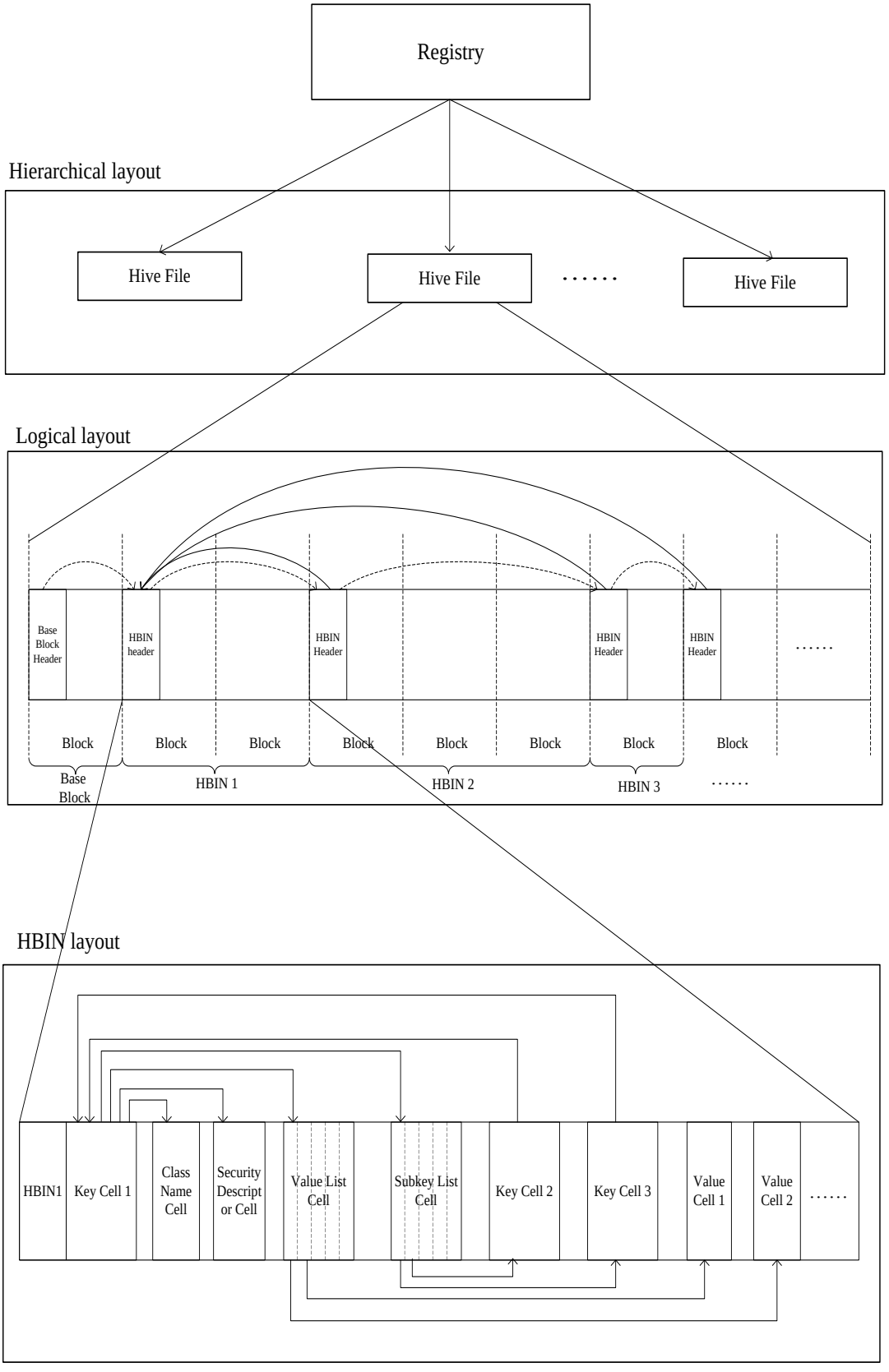


Figure 4.2 The logical layout of Windows registry

From the logical layout, we can easily know the hive file internal reference information can be used to recover deleted or lost hive files. Moreover, the HBIN layout provides us further sufficient information to help us cross the balk when we recover the fragmented hive file. In the following, we will explain the further details inside the HBIN.

### 4.1.2 HBIN Structure

#### 4.1.2.1 HBIN Header Data Structure

Different from base block, the bin header always has 0x20 bytes which includes the HBIN signature “hbin”, the offset to the first HBIN, the size of this HBIN and the timestamp, shown in Figure 4.3. The offset from each HBIN to the first HBIN indicates the HBIN’s logical location in this hive file. The bin size can be interpreted as an offset to the next bin. In other words, bins are chained together by the bin size in that each bin points to the bin that follows. The timestamp in the first HBIN is the same as the hive file header; unfortunately, we can only find the first HBIN has the same timestamp as the base block, while the rest of HBINs are set to 0x00000000. In summary, the offset to the first HBIN and the size of the HBIN give us the important clues to recover the fragmented hive files.

| hbin singature | Offset to the 1st HBIN | Bin size | Reserved | Timestamp | Reserved |
|---|---|---|---|---|---|
| 0x04Bytes | 0x04Bytes | 0x04Bytes | 0x08Bytes | 0x08Bytes | 0x04Bytes |

Figure 4.3 HBIN header data structure

Here is an example from a DEFAULT hive file, Figure 4.4, which shows the relationship between HBINs. This DEFAULT file has 5 fragments and every HBIN occupies one cluster where the cluster size is 4096 bytes. In the first fragment, after the base block, the 1st HBIN is located at cluster 204397 and the last cluster of this fragment is 204451 as well as it represents the 55th HBIN. The offset to the first HBIN in 55th HBIN is 0x36000 bytes. In the second fragment, it starts at cluster 18976 which is located in front of the first fragment on hard disk and it is the 56th HBIN of the hive file. The offset to the first HBIN in this HBIN is 0x37000 bytes. If we do the proof calculation, 0x36000 and 0x37000 divided by HBIN size 0x1000, separately, they are 54 and 55. Plus the first HBIN, they are the 55th and 56th HBIN in the HBIN order. Furthermore, from this example, we can know that the fragment can be stored out of order on the hard disk and the offset to the first HBIN is the absolute value in bytes to the first HBIN.

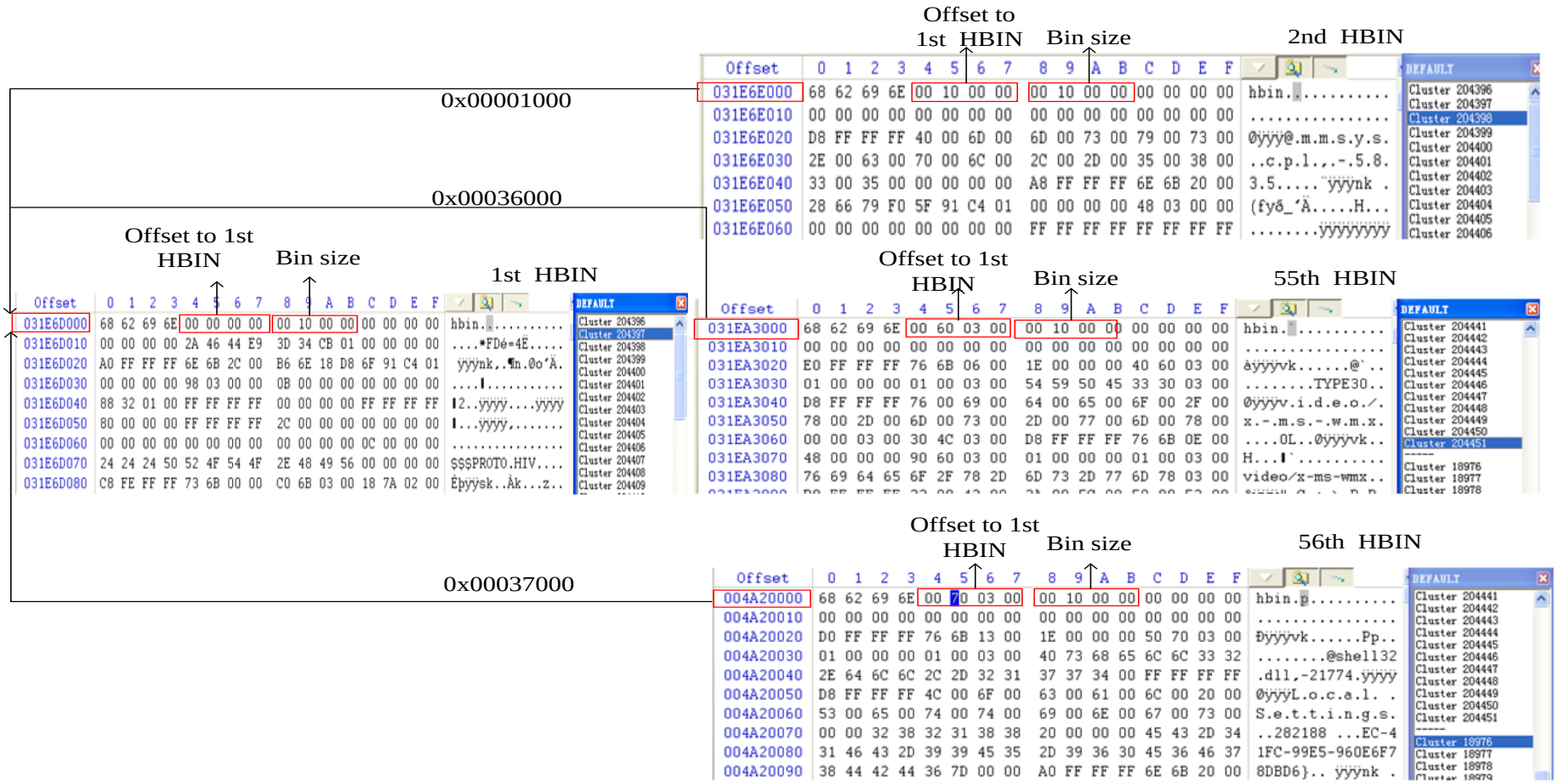


Figure 4.4 The relationship between HBINs

#### 4.1.2.2 HBIN Data Structure

Besides the HBIN header, the rest of HBIN is made up by reference related cells. The cell, which is the unit to store the data in HBIN, is variable-length but is a multiple of 8 bytes. A hive file has a number of different cells: key cells, sub-key list cells, value list cells, value cells and the security description cell. Table 4.1 shows the signature of different cells. We will detail the cell information in the sub-sections.

Table 4.1 The signature of different cells

| Cell Names | Signatures |
|---|---|
| Key cell | nk |
| Sub-key list cell | lf/lh, ri/li |
| Value cell | vk |
| Security descriptor cell | sk |

In each hive file, the reference point is the first HBIN offset value. In each HBIN, all cells list out the necessary related cells offset based on the reference point. According to the offset and reference point, the system can easily locate and extract the information which requires access rights for the user [42]. Figure 4.5 shows the data structure of an HBIN.

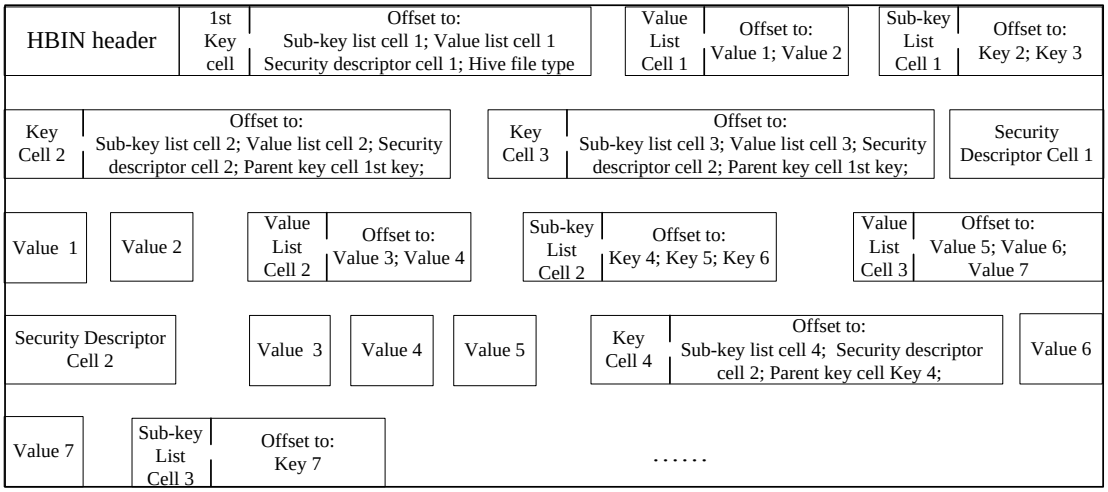


Figure 4.5 The data structure of HBIN

Next, we will introduce several important data structures in the HBIN.

a. Key Cells

The key cell plays a significant role in the hive file structure which contains all the reference information of the cell and ties all cells together. Each hive file contains a registry tree, which has a key that serves as the root of the tree, so people call it the root key cell. Actually, the regedit.exe shows the relationship between the hive file and keys are not straightforward. In other words, the root keys do not exist on the hard disk [24]. But from the survey, we found that there is a key cell always following the first HBIN header in hive files. We call it the 1st key cell. Then the system constructs the hive file from the 1st key cell. In particular, the key cells record the offset to the sub-key list cells, the value list cells and the security description cell. In addition, the key cells record the reference to their parent key cells. Figure 4.6 is an example of a

key cell and shows the lengths of the above-mentioned cells and offsets in the key cell.

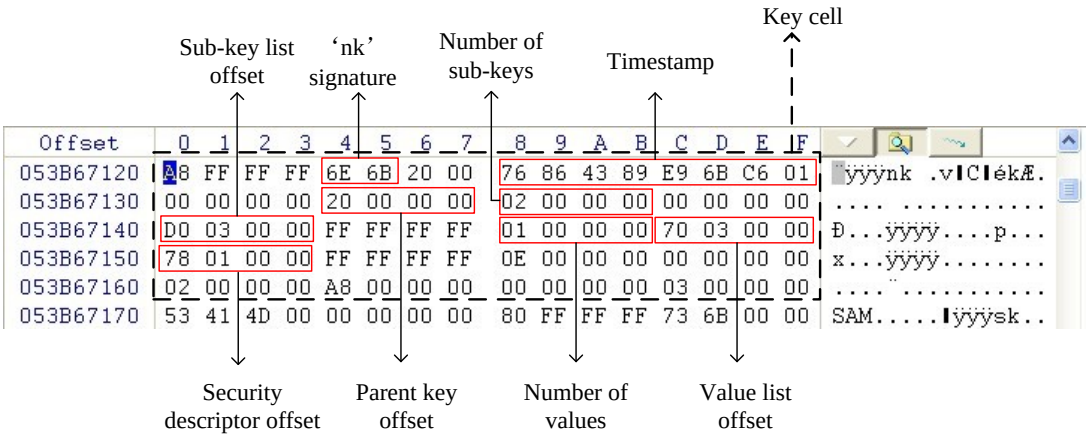


Figure 4.6 The important values in the key cell

From the 1st key cell, and according to the reference to sub-key list, we can draw out the whole registry tree of this hive file. Figure 4.7 shows an example of key cell relationships - how the key cells are connected together by the sub-key offsets in the sub-key lists and the offset to the parent key which is part of a SAM hive. Clearly, the key cell at 0x53B692F0 references the sub-key list at offset 0x2658, and based on the first HBIN offset 0x53B67000, you can find out the sub-key list at offset 0X53B69658 which lists the offset to three key cells. When you look into any one of the three sub-keys, you can find at the field of the parent key offset that the offset value references the key at 0x53B692F0. The hive file is easily organized by the same strategy.

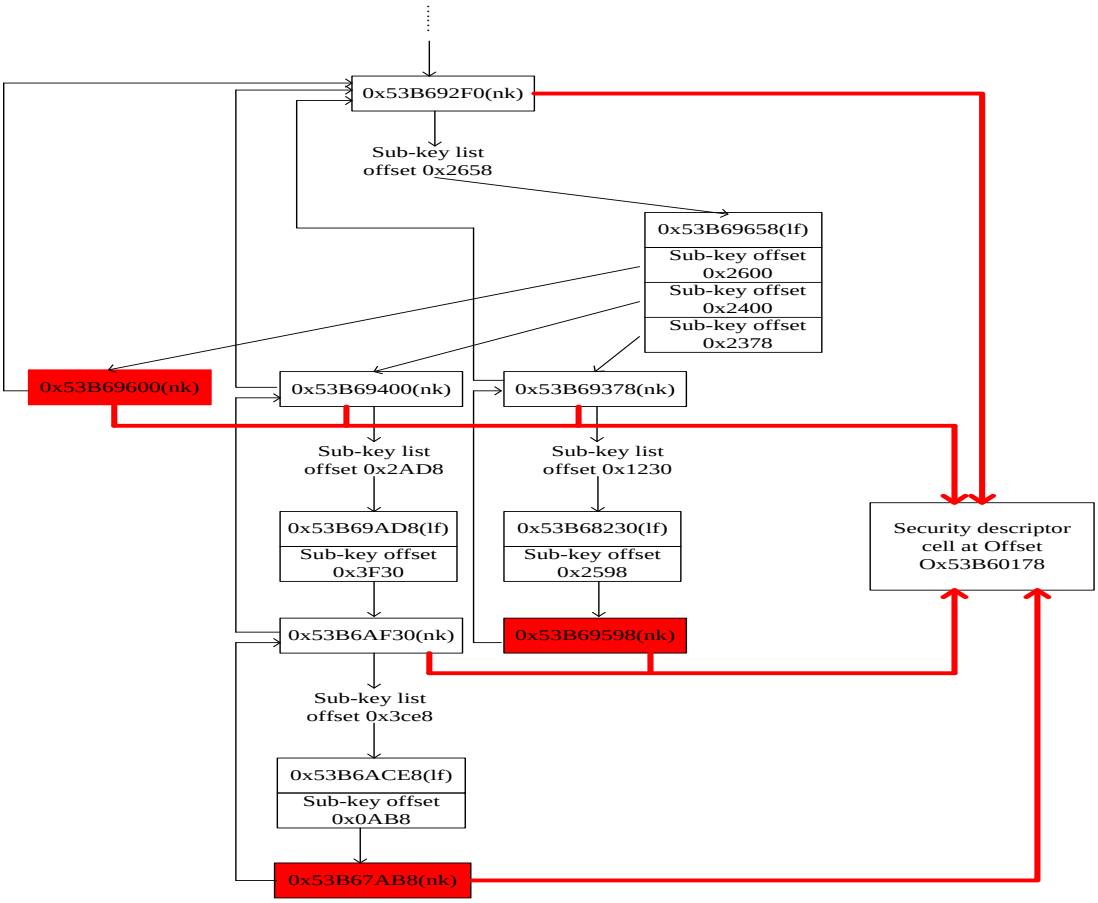


Figure 4.7 Key cell relationships

But when a key does not have a sub-key, the field of the number of sub-keys will set to 0x00000000 and the sub-key list offset will set to 0xffffffff. In Figure 4.7, the key cells are in red color if they do not have further sub-keys. The field for a number of values and value list offset will follow the same rules. At the end of the day, when the sub-key list offset is set to 0xffffffff and number of sub-keys is set to 0x00000000 in each key cell, it means this HIVE file ends.

b. Sub-key list Cells and Value-list Cells

In order to search, locate and extract the sub-keys quickly and correctly, the key cell records the sub-key list cell which is sorted in order by the hash value of the cell name. Then, the key cell references the sub-key cells through the sub-key list. However, the key cell organized is not in a sequential order which means the sub-keys can exist in front of their parent key on the hard disk. The value-lists are similar to the sub-key lists which reference value records, but the value-lists are not sorted in order.

c. Security Descriptor Cells

The structure of security descriptor cell contains the security information associated with an object. The cell records include a short header followed by a Windows security descriptor which defines permission and ownership for local values or sub-keys. Applications use this structure to set and query an object's security status. Multiple key cells can share the same security descriptor cells. In Figure 7, following the red strength line, all the ‘nk’ signed cells in the image reference the security descriptor cell at offset 0x53B60178. Actually, in this hive file, there are only two security descriptor cells which the 1$^{st}$ key cell has one and the rest of the key cells share another one.

## 4.2 Fragmentation in Windows Hive File

In this section, we study fragmentation issues in Windows Hive Files through a study, which was based on a survey of 50 Windows systems with the NTFS (New Technology File System) file system, where NTFS is the standard file system of Windows systems.

The registry is the most frequently updated in the Windows system in that it holds all the configuration data about the hardware devices and software programs. As a result, it is not unusual to see that the Windows registry files are fragmented. Theoretically, the fragmentation can happen anywhere in the hive files. But according to the Microsoft system storage strategy, the data are filled in the cluster logically. According to the drive size nowadays, the cluster size is bigger than the hive block size [43]. In other words, the fragmentation can just happen at the end of a cluster.

First, we introduce a study conducted by us and present statistics about the incidence of file fragmentation on actual file systems in which we try to summarize some characteristics to help us recover the hive file. We investigated 50 systems in total, 20 of which were Windows XP, 12_systems were Windows Vistas, 13 systems were Windows 7 and the remaining 5 were Windows 2003 servers and Windows 2000. Table 4.2 shows the fragmentation in different systems from the survey.

Table 4.2 The fragmentation in different systems

| | Total System | With Fragmentation | Without Fragmentation | Fragmentation Rate |
|---|---|---|---|---|
| Windows XP | 20 | 13 | 7 | 65% |
| Windows 2003 Server/2000 | 5 | 5 | N/A | 100% |
| Windows Vista | 12 | 5 | 7 | 42% |
| Windows 7 | 13 | 3 | 10 | 23% |

From table 4.3, we can know that most of the fragmentation happen in software and system hive file, because these hive files are always deleted or changed by the user. The other files tend to be the same as they are initiated when the file was created by Microsoft.

Table 4.3 The fragmentation in different hive files

| | Software | System | Dfault | Ntuser | SAM | UsrClass |
|---|---|---|---|---|---|---|
| Windows XP | 11 | 4 | 4 | 4 | 1 | 1 |
| Windows 2003 Server/2000 | 4 | 3 | 2 | 1 | N/A | N/A |
| Windows Vista | 3 | 3 | N/A | N/A | N/A | N/A |
| Windows 7 | 3 | 1 | N/A | N/A | N/A | N/A |

According to the survey, Windows Registry files could be fragmented into as many as one hundred pieces which make recovery of deleted/lost registry file extremely hard. Moreover, these fragments are stored out of order in the hard disk, which increases the difficulty of recovery. In addition, most of the HBINs are one cluster, but there are a number of HBINs made up by several clusters. Theoretically, the fragmentation

happens at the end of the cluster [22, 43]. From our survey, we find that the fragmentation do as happen at the end of the cluster. However, in Windows registry hive files, when the HBIN size equals the cluster size, the fragmentation happens at the end of the cluster or HBIN, but when the HBIN size occupies over two clusters, the fragmentation can happen in the middle of HBIN which is at the end of the cluster.

1) Scenario A: The fragmentation happens at the end of previous HBIN (Figure 4.8). Most of cases are under this situation, and we can easily infer the fragmentation point based on the HBIN offset to the first HBIN in this hive file.

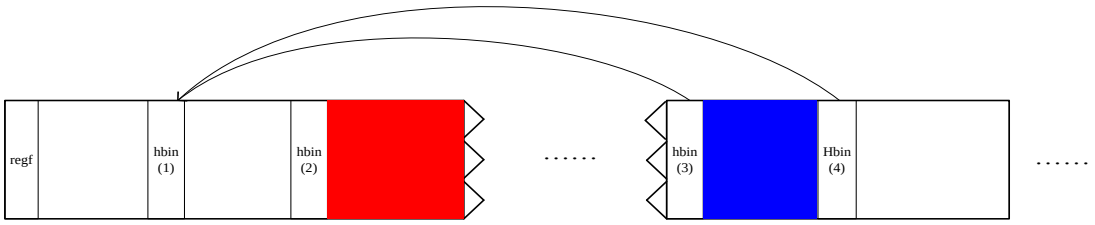


Figure 4.8 The fragmentation between HBINs

2) Scenario B: In rare cases, the fragmentation is in the middle of one HBIN (Figure 4.9). In this case, except the HBIN header that has the information which is the offset to the first HBIN and the HBIN size, the rest of block in this HBIN are just filled with data. In other words, we cannot deduce the fragmentation point according to previously mentioned characters. What we can do is look for the fragmentation point and the starting cluster according to the relationship among the key cells.

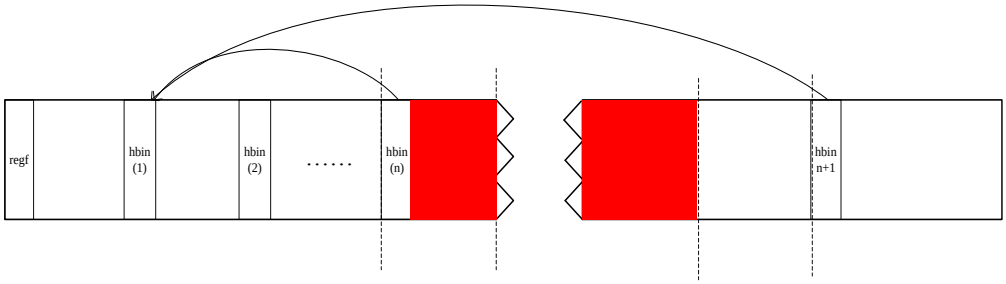


Figure 4.9 The fragmentation in HBIN

Moreover, in most fragmented hive files, the first piece is always very big. We suppose the system initially allocates a big space for each hive file, but as the operation unfolds the space is not enough for holding the data. Since operating on the system requires loading new information and data, the system tries to allocate some new space for the file. These spaces are distributed arbitrarily on the hard disk, but sizes are quite small and most of which are no more than 100 clusters and tend to be the same. In other words, the system tries to allocate the same space for the following fragments. From the survey, we can know 48% of the operating system hive files do not have a single fragmented file; 52% of the system's hive files have various fragments from one to hundreds. Table 4.4 shows that most of the fragments were found to be between 1 to 64 clusters.

Table 4.4 The fragment size

| Fragment size(clusters) | # Fragment |
|---|---|
| 1 -- 32 | 109 |
| 33 -- 64 | 298 |
| 65 -- 96 | 41 |
| 97 -- 128 | 11 |
| 129 -- ∞ | 14 |

According to the survey results and the internal structure of the hive file, we can design the following algorithm for recovering the fragmented hive files. An effective file carving algorithm is based on Windows registry files' internal structure, which will be detailed in the next sub-section.

## 4.3 File Structure based Carving Algorithm for Windows Registry

In this section, we propose an effective carving algorithm for Windows registry files based on the internal structure and unique patterns of storage concluded from our study.

Figure 4.10 shows the main flow of the algorithm. In the following, we will detail the algorithm step by step and show how to carve the deleted/lost Windows registry hive files.

```
Initial
For i  0 to n do lookup "regf"                    // n is the total clusters
      LR: For j  0 to m do
            If  S[j] + F[j] = F[j+1]   then       // S[j] is the size of HBIN[j]
                                                  //  F[j] is the offset of HBIN[j] to 1stHBIN
                                                  // F[j+1] is the offset of HBIN[j+1] to 1stHBIN
                        Two clusters are contiguous
                        Return the hive file is contiguous
             Else
                        The hive file is fragmented between HBIN F[j] and F[j+1]
             If HBIN F[j] is a complete HBIN
                 For k  0 to n do lookup for HBIN F[j+1]
                  If HBIN F[k]  = F[j+1]  then
                          HBIN F[j+1] is the candidate, Return to R&V
                   Else this fragment can be overwritten or damaged
                          Jump to Initial
           Else
                For p  0 to n search the key cell at set offset in the HBIN
                  If NK in HBIN F[k] +/- offset from Sub-key list = Nk in HBIN F[k+1]
                          It is the candidate of HBIN F[n+1], Return to R&V
                   Else this fragment can be overwritten or damaged
                          Jump to Initial
      R&V : Reassemble HBIN F[j] & F[j+1]
            If the HIVE file is a complete, return to Initial
            Else Return to LR
End
```

Figure 4.10 File structure based file carving algorithm

Step 1: Find the Base Block of the Hive File

Scan the disk image, and if a cluster begins with a signature "regf" and the file name is one of the hive files, this cluster is the base block of a hive file.

Step 2: Check Whether the Hive File is Contiguous or Fragmented

Normally, the HBINs have the magic number "hbin" and all the HBINs will follow the base block. Every HBIN stores their size $S_{(n)}$ in the header. Based on this structure, we can use the offset of HBIN at $H_{[n+1]}$ to minus the offset of HBIN at $H_{[n]}$, so that we can know the size of HBIN $O_{(l)}$ at $H_{[n]}$.

$$\begin{cases} O_{(l)} = O_{(H[n+1])} - O_{(H[n])} & (4.1) \\ O_{(l)} = S_{(n)} & (4.2) \end{cases}$$

In other words, if offset and the HBIN size satisfy the equations 4.1 and 4.2, these two HBINs are contiguous, shows in Figure 4.11 Repeat the same way, we can know whether this hive file is fragmented.

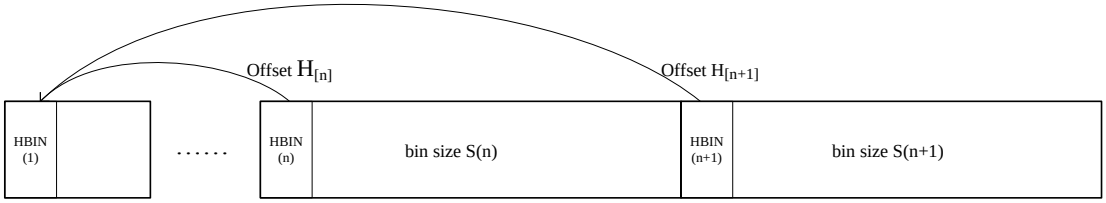


Figure 4.11 The contiguous HBINs

For the intact hive file, it will be easy for us to extract the data and recover the file. But for the fragmented hive file, it is difficult to locate the fragmentation and to recover the hive file, because from the survey we know that these fragments are distributed arbitrarily on the hard disk. The next step is to identify the ending cluster of the fragmentation.

Step 3: Identify the Fragmentation Point

We need to consider two situations when we try to find out the fragmentation point. According to the most common clusters in one fragment, we can first search for those

frequently fragmented cluster numbers. After this, we can check the results in the following conditions.

Firstly, the fragmentation happens at the end of one HBIN. If the hive file is fragmented, according to the HBIN structure, we can know the size of this HBIN. Moreover, the last cell of a HBIN fills out the remaining space of a bin. In other words, there is not any slack space in a HBIN [27]. Then, according to the size of the HBIN, we can trim this fragment out at the end of this HBIN to which this HBIN size offset is pointed. Then, we can extract this fragment out.

Secondly, the fragmentation happens between the clusters in HBIN. If the hive file is fragmented in the middle of one HBIN which is made up by several clusters, we cannot locate the fragmentation point through the connection between the HBINs. However, the key cell contains information which references the parent key cell and the sub-key cells distributed in all HBINs of one hive file. If the reference between the key cells has no relationship, we can know the fragmentation happens in the middle of the two HBINs.

Step 4: Find the Beginning of Next Fragment Beginning Block

To identify the fragmentation point, we also need to consider two situations when we are looking for the starting cluster of the following fragment.

Firstly, the fragmentation happens at the end of one HBIN. If we find the fragmentation point of previous fragment, the next fragment of this file will start with the signature “hbin”. Moreover, the offset to the first HBIN is sequential following the previous offset. Hence we can scan the whole disk and try to find the offset value which is matched with the offset for the next HBIN. It is possible to find more than one HBIN with the same offset, but we can eliminate those in the sequence and keep those which are not following the previous HBIN.

Secondly, the fragmentation happens between the clusters in HBIN. When the fragment point of the previous fragment is found, the following fragment point of this file will contain a key cell which the offset to the parent or sub-key list of the key must relative to the previous key cell. According to this, we can search the starting cluster from the disk.

In order to keep a low false positive, we introduce the validation algorithm to make sure the fragments that we found belong to the same file.

Step 5: Reassembly and Validation

Reassemble those extracted fragment together, and if the field of the number of sub-keys in all ending key cells are 0x00000000 and the sub-key list offsets are 0xffffffff, it means there are no more fragments for this hive file. Validation is the

way to make sure all the fragments which we found belong to the one hive file. We can reconstruct the tree of this hive file. If all the references among the HBIN header and the key cells do not have a single break, this hive file is the right one.

### 4.4 Discussions

In this section we will discuss some special cases such as the cell size over HBIN size, several HBINs missed and a potential clue to help us recover the deleted/lost Windows registry hive files.

#### 4.4.1 Cell Size over HBIN Size

If a value cell size is bigger than the block size and it takes more than two blocks in a HBIN, the fragmentation happens in the middle of the value cell [44]. In Figure 4.12, for example, the grey colored value cell starts from the 1$^{st}$ block until the beginning of the 3$^{rd}$ block. If the fragmentation happens at the end of the 1$^{st}$ block, we cannot locate the fragment point according to the algorithm in that we cannot get any helpful information from the 2$^{nd}$ block.

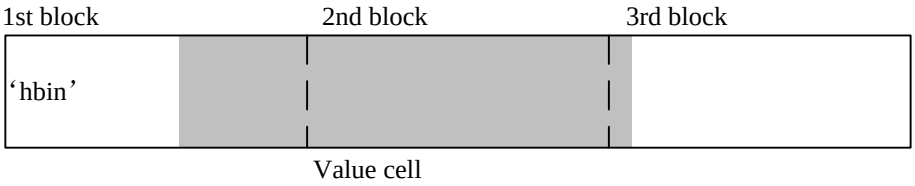


Figure 4.12 The cell size is bigger than one block

### 4.4.2 Missing HBINs

We can recover some HBINs, but few HBINs cannot be allocated in that those HBINs may be overwritten or damaged by the system. In this case, we can set a threshold value according to the file size or HBIN quantity. If the extracted HBINs over a set value or threshold value, we can take it as a complete file when we can get value information from it.

### 4.4.3 A Potential Clue: Security Descriptor Cell

Security descriptor cell can be used for looking up the fragment or validation in that multiple key cells can share the same security descriptor cell. But at the beginning of the survey, we did not think about recording how the security descriptor cell relates to the key cells. It can be another clue like in the example SAM hive file, which all key cells reference the same security descriptor cell.

# Chapter 5 Conclusions and Future Work

## 5.1 Summary of Research Work

In our research, we realize that computer forensics provides forensic accounting services, conducts fraud investigations, and assists counseling in litigating civil and criminal cases based on locating resident, hidden, or deleted data from the computer. File carving as a technique is a Swiss Army Knife in computer forensics which can help us overcome the problems which cannot be handled by those traditional data based on file system metadata recovery tools. Even though many related works have been done on carving different files in distinct situations, lots of problems still exist and need to be solved. Based on related work, we conclude the main problems during file carving and some drawbacks in these approaches. In order to solve the practical problems, we propose our designs to supplement a famous forensic tool COFEE to investigate unallocated space on the hard disk at the crime scene and introduce a way to carve the Windows registry hive files to help investigators extract more useful information.

Specifically, in our research, we propose a new approach to store large amounts of data in a space efficient data structure, Bloom filter, for easy portability. At the same time,

we use hash trees to efficiently query the existence of files of interest in the Bloom filter, and apply group testing to detect the potential fragmentation point of the file. The gap distribution statistics between the file fragments was further applied to narrow down the region which was searched for the starting cluster of the next fragment. This approach allows us to quickly query for relevant files from the suspect's device during a 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.

As a secondary investigation, we propose an approach to carve the registry hive file without any system metadata when the hive file is either contiguous or fragmented. From related papers and tutorials, we clarify part of the interior structure of the hive file. Due to the limited information of the Windows registry hive file, we did a survey from 50 computers to verify previous works and most importantly, we supplemented more details to construe the structure and the fragment characteristics based on the statistics we gathered from the survey. Based on a brute force search algorithm, we can accurately extract the registry hive files which offer forensic specialists a whole picture of the Windows registry and help them efficiently locate the information that is likely to be of interest.

## 5.2 Summary of Key Outcomes and Contributions

Specifically, the contributions of the research in the efficient evidence extracting method 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 along with statistics about the size of gaps between two fragments of a file for effectively searching the unallocated space of the suspect device to extract fragmented files that were permanently deleted. The contributions in the novel file carving algorithm for Windows registry files lie in: First, we did a survey based on 50 computers installed different systems to verify hive file structure and we supplemented more new discovered details to clarify the structure and the fragment characteristics based on the statistic from the survey to help us get a more accurate recovery result. Based on the survey result, we propose a novel file carving algorithm for Windows registry files.

## 5.3 Recommendations and Future Research

Computer forensics use the updated technology to analyze, glean and demonstrate the evidence needed for law enforcement officers in civil or criminal courts to convict those suspected crimes. In this thesis, we proposed an efficient evidence extracting method from unallocated space which is based on hash operation. The main drawback of hash based methods which detect similar files is that files of investigative interest cannot be modified. It is highly possible that criminals may manipulate improper files to avoid being caught by using tools introduced in this work. In the future, we plan to address this issue by adopting fuzzing-hashing function [45, 46, 47].

Furthermore, in computer forensics, file carving is an essential tool which provides the fundamental technology to help the forensic specialist extract the wanted data. In the future, we will develop a universal framework which can be used to carve different types of files such as wav, mp3, xml from unallocated space based on their own unique internal structure so that we can facilitate analysis and extract various files from unallocated disk space.

**Bibliography**


[1]. X. Lin, C. Zhang, and T. Dule. "On Achieving Encrypted File Recovery", in Proc. 3rd International ICST Conference on Forensic Applications and Techniques in Telecommunications, Information and Multimedia, November, 2010.

[2]. "THE ENRON SCANDAL", Internet: http://whatreallyhappened.com/WRHARTICLES/enron.html, [Jun. 24, 2011].

[3]. "Computer Online Forensic Evidence Extractor", Internet: http://www.microsoft.com/industry/government/solutions/cofee/default.aspx, [Jun. 24, 2011].

[4]. "Computer Online Forensic Evidence Extractor", Internet: http://www.microsoft.com/industry/government/solutions/cofee/default.aspx, [Nov. 12, 2009].

[5]. "COFEE Microsoft Forensic Tool Download", Internet: http://digiex.net/downloads/download-center-2-0/applications/2812-cofee-microsoft-forensic-tool-download.html, [Jun. 24, 2011].

[6]. M. Blanchfield. "Roman Catholic Bishop Raymond Lahey pleads guilty to child-porn charge", Internet: http://www.theinquiry.ca/wordpress/charged/bishop-raymond-lahey/roman-catholic-bishop-raymond-lahey-pleads-guilty-to-child-porn-charge/, May 04, 2011[Jun. 24, 2011].

[7]. A. Broder and M. Mitzenmacher. "Network Applications of Bloom Filters: A Survey", Internet: http://www.eecs.harvard.edu/~michaelm/NEWWORK/postscripts/BloomFilterSurvey.pdf, 2010 [Dec. 1, 2009].

[8]. B. Carrier. File System Forensic Analysis. Publisher: Addison Wesley Professional, 2005, pp. 156-168.

[9]. S. L. Garfinkel. "Carving contiguous and fragmented files with fast object validation", Digital Investigation. vol. 4, suppl. 1, pp. 2-12, 2007.

[10]. "Computer forensics", Internet: http://en.wikipedia.org/wiki/Computer_forensics, January 2011 [Jan. 28, 2011].

[11]. Computer Forensics. (n.d.). "An introduction to: Computer Forensics", Internet: http://www.secureworks.com/assets/uk/ComputerForensics.pdf, [Oct.2, 2009].

[12]. Encase. Available: http://www.guidancesoftware.com/, [Oct.2, 2009].

[13]. WinHex. Available: http://www.winhex.com/winhex/,[Oct.2, 2009].

[14]. Forensic Toolkit (FTK). Available: http://accessdata.com/products/forensic-investigation/ftk,[Oct.2, 2009].

[15]. E. Casey. "Tool Review – WinHex", Digital Investigation. vol.1, iss. 2, pp. 114-128, Jun. 2004.

[16]. Foremost. Available: http://sourceforge.net/projects/foremost/, [Oct.4, 2009].

[17]. Scalpel. Available: http://www.digitalforensicssolutions.com/Scalpel, [Oct.4 2009].
[18]. G. G. Richard Ⅲ and V. Roussev. "Scalpel: A frugal, high performance file carver"， in Proc. Digital Forensic Research Workshop, 2005.
[19]. Y. Wei, N. Zheng, and M. Xu. "An Automatic Carving Method for RAR File Based on Content and Structure", in Second International Conference on Information Technology and Computer Science, 2010, pp.68-72.
[20]. M. Chen, N. Zheng, X. Xu, Y.J. Lou, and X. Wang. "Validation Algorithms Based on Content Characters and Internal Structure: The PDF File Carving Method", in Proc. International Symposium on Information Science and Engineering, Dec. 2008, vol. 1, pp.168-172.
[21]. N. Menom and A. Pal. "Automated reassembly of file fragmented images using greedy algorithms", IEEE Transactions on Image Processing, vol.15, no. 3, pp. 385 – 393, Mar. 2006.
[22]. A. Pal, H.T. Sencar, and N. Menom. "Detecting file fragmentation point using sequential hypothesis testing", in Proc. the Eighth Annual DFRWS Conference, Aug. 2008, vol. 5, suppl. 1, pp. S2-S13.
[23]. H.T. Sencar, and N. Menom. "Identification and recovery of JPEG files with missing fragments", in Proc. the Ninth Annual DFRWS Conference, Sep. 2009, vol. 6, suppl. 1, pp.S88-S98.
[24]. M. Russinovich. "Inside the Registry", Internet: http://technet.microsoft.com/en-us/library/cc750583.aspx, [Sep. 11, 2010].
[25]. T.D. Morgan. "The Windows NT* Registry File Format Version 0.4", Internet: http://www.sentinelchicken.com/research/registry_format, Jun., 2009, [Sep. 11, 2010].
[26]. T.D. Morgan. "Recovering deleted data from the Windows registry", in Proc. the Eighth Annual DFRWS Conference, Sep., 2008, vol. 5, suppl. 1, pp. S33-S41.
[27]. J. Thomassen. "Forensic analysis of unallocated space in WINDOWS registry hive files", Master's thesis, The University of Liverpool, England, Apr. 2008.
[28]. W.C. Calhoun and D. Coles. "Predicting the types of file fragments", in Proc. the Tenth Annual DFRWS Conference, 2010, vol. 7, suppl. 1, pp. S24-S31.
[29]. C.J. Veenman. "Statistical Disk Cluster Classification for File Carving", in Proc. the Third International Symposium on Information Assurance and Security, Aug. 2007, pp.393-398.
[30]. C. Antognini. "Bloom Filters", Internet: http://antognini.ch/papers/BloomFilters20080620.pdf, 2008 [Dec., 2009].
[31]. L. Fan, P. Cao, J. Almeida and A. Broder. "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol", in Proc. ACM SIGCOMM'98, 1998.
[32]. "Squid: Optimising Web Delivery", Internet: http://www.squid-cache.org/, [Apr., 2011].
[33]. A. Broder and M. Mitzenmacher. "Network Applications of Bloom Filters: A Survey", Internet:

http://www.eecs.harvard.edu/~michaelm/NEWWORK/postscripts/BloomFilterSurvey.pdf, 2002 [Nov., 2009].

[34]. N. Hua, H. Zhao, B. Lin, and J. Xu. "Rank-Indexed Hashing: A Compact Construction of Bloom Filters and Variants", in Proc. IEEE Conference on Network Protocols (ICNP), Oct. 2008, pp.73-82.

[35]. B. Carrier. File System Forensic Analysis. Publisher: Addison Wesley Professional, Boston, Massachusetts, Mar. 2005, pp.156-184.

[36]. D. Wei. "Speed Comparison of Popular Crypto Algorithms", Internet: http://www.cryptopp.com/benchmarks.html, Mar. 31, 2009 [Apr. 25, 2011].

[37]. Y. Hong and A. Scaglione. "Generalized group testing for retrieg distributed information", in Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Mar., 2005.

[38]. J. Chapweske and G. Mohr. "Tree Hash Exchange format (THEX)", Internet: http://zgp.org/pipermail/p2p-hackers/2002-June/000621.html, Jun., 2002 [Jan. 11, 2010].

[39]. F. Hwang, and D. Du. Combinatorial Group Testing and Its applications (2nd edition). Publisher: World Scientific, Singapore, 1999, pp. 13-37.

[40]. G. M. Zaverucha, and D. R. Stinson. "Group Testing and Batch Verification", in Proc. the 4th international conference on Information theoretic security, 2009 (ICITS'09).

[41]. "Registry Hives (Windows)", Internet: http://msdn.microsoft.com/en-us/library/ms724877%28v=VS.85%29.aspx, [Jun. 6, 2010].

[42]. "Retrieving Data from the Registry (Windows)", Internet: http://msdn.microsoft.com/en-us/library/ms724925%28v=VS.85%29.aspx, [Jun. 6, 2010].

[43]. "I-50970126 - Windows NT Default Cluster Size for FAT and NTFS", Internet: https://support.intergraph.com/itservices/iarticles/50970126.pdf, [Jun. 7, 2010].

[44]. "Undocumented Windows Vista and later registry secrets", Internet: http://www.msuiche.net/2009/06/07/windows-vista-and-later-registry-secrets/, [Jun. 7, 2010].

[45]. J. Kornblum. "Identifying almost identical files using context triggered piecewise hashing", in Proc. The Digital Forensic Workshop, August 2006, pp. 91-97.

[46]. D. Hurlbut. "Fuzzy Hashing for Digital Forensic Investigators/AccessData", Internet: http://accessdata.com/downloads/media/Fuzzy_Hashing_for_Investigators.pdf, Jan.9, 2009 [May 27, 2011].

[47]. K. Seo, K. Lim, J. Choi, K. Chang, and S. Lee. "Detecting Similar Files Based on Hash and Statistical Analysis for Digital Forensic Investigation", in Computer Science and its Applications, 2009, pp. S.1-6.