EFFICIENT N-BYTE SLACK SPACE HASHING IN RETRIEVING AND IDENTIFYING PARTIALLY RECOVERED DATA

Ireneusz Jozwiak and Michal Kedziora
Institute of Informatics, Wroclaw Univesrity of Technology, ul. Wybrzeże Wyspianskiego 27, 50-370, Wroclaw, Poland

Keywords: Data retrieval, e-Discovery, Hash functions, Data recovery.

Abstract: This paper describes modification of slack space block hashing algorithm which improves performance in the handling with process of identification of recovered data. In our research we relied on hash block algorithm and present improvements which allow increase efficiency by analyzing time reduction. N-Byte Slack Space Hashing is especially useful in data recovery process where due to a file system limitations, it is possible to recover only fragments of data which was erased and partially overwritten. The Algorithm is faster than block hashing and allows to identify partially erased files using modified hash sets.

1 INTRODUCTION

One of major tasks during data analysis process is to search and identify specific files. This task is even more complicated when applying data retrieving techniques to applications as data recovery and computer forensics. The simplest method is to compare files by their names and extensions (Kornblum, 2006). Obviously this method is highly inefficient and can be easily compromised by changing files names (Bunting, 2008). Most forensic analysis software tools (Casey, 2004) can detect that someone change file extension to hide evidence by performing file signature analysis. In this process file extension is compared with its header. Performing name search and file signature analysis is one of basic steps in computer forensic investigations but there are not efficient way of searching and identifying data. More advanced method is to made search using one way cryptographic hash functions and specially created hash tables of known files (Henson, 2003). One of the biggest advantages of comparing files by hash is that it gives positive results even if file name was changed because hash value is computed on a data part of the file, and not on directory entry which keeps the name and other metadata information (White, 2005). Condition which has to be fulfilled to properly use hash analysis is possessing whole data of logical file to create its hash value (Stein, 2005). Unfortunately the most interesting files are often deleted both accidently and intentionally. For example Internet activity history is usually deleted by internet browser after fixed time period without being noticed by the user. On the other hand we can have situation where the user intentionally deletes incriminating data. Hopefully it is not so easy to completely delete data from the hard drive. In simple, due to efficiency issues, when a file is deleted from computer it is only simply removed from a directory of files (Microsoft, 2004). It's content still exist intact but an operation system marks space as unallocated. Because the OS doesn't immediately re-use unallocated space from deleted files, a file can be recovered right after it has been erased, and for a considerable time afterwards. Chances of a whole logical file recovery decrease with time, it is because sooner or later some or all of that unallocated space will be re-used. Fortunately in most cases not all unallocated space is overwritten and we can recover part of previously deleted data (Breeuwsma, 2007). This is case we will deal in this work. We are not able to recover whole file so we cannot compute its hash, and compare it with hash table. Solution is to make several hashes from each file to make comparison possible. Selection of the proper algorithm we precede with analyzing slack space forming and creating mathematical model of data recovery process.

Jozwiak I. and Kedziora M..
EFFICIENT N-BYTE SLACK SPACE HASHING IN RETRIEVING AND IDENTIFYING PARTIALLY RECOVERED DATA.
DOI: 10.5220/0003605703090312
In *Proceedings of the 6th International Conference on Software and Database Technologies* (ICSOFT-2011), pages 309-312
ISBN: 978-989-8425-76-8

2 THEORY

The file system is essential to store and organize computer data. It can be described of as an index or database containing the physical and logical location of every piece of data on a hard drive. Most popular nowadays are two file system types FAT and NTFS. The basic concept of a FAT file system is that each file and directory on a disk is allocated into a data structure (Microsoft, 2000), which is called a directory entry. It contains the file size, name, starting address, and other metadata. The file and the directory content is stored in data units called clusters (standard is 8192 bytes for one cluster, but it can be defined differently). If the file or directory has allocated more than one cluster, the other clusters are found by using a structure that is called the FAT. Next popular file system is NTFS. NTFS core is Master File Table (MFT). It holds the information about every file and directory located on the drive (Berghel, 2007). Important from our point of view are data units allocation strategies, used to place file content into disk. In the best case, operation system should allocate consecutive data units, but that is not always possible. In the course of time files are put in and out from drive, allocating large files in one piece may become a problem. When a file does not have consecutive data units, it is called fragmented An operation system can use several different strategies for allocating data units.

2.1 File System Slack Space

Most popular file systems (FAT 32, NTFS) have structure based on blocks (called also sectors) which have standard data length 512 bytes and clusters representing 8 sectors space. The smallest space allocated for file is one cluster. This mean that even though file size is 10 bytes there will be always 8192 bytes cluster reserved for it. We trace this process according to figure 1 example.

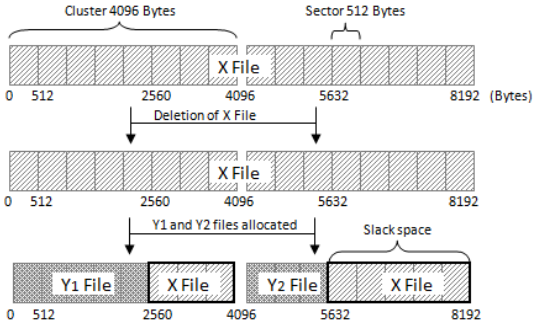

Figure 1: Formation of the slack space.

Figure 1 shows scenario with 8192 Bytes X File which occupies two whole clusters. The X file has been deleted. During this process entries of the X file are deleted (process can be slightly different depending on the file system used).Process will not affect any data stored in clusters, all data is still available but operation system will mark the space as unallocated and ready to reuse. The next phase of this scenario is when two files Y1 and Y2 are created. First has 2560 Bytes logical size, second 1536 Bytes. According to the file system allocation strategy, each file will be placed in the beginning of next available cluster. File Y1 will start in byte offset 0 and end in 2560. Next cluster begins in offset 4096, therefore the slack space will be created between offset 2560 and 4096. Analogous situation will be with Y2 which starts in offset 4096 to 5632. Because cluster size is 4096 byte, slack space size of 2560 bytes is created. If X File where valuable file from forensic view it would be good to identify it. Slack space size can have different value, but in this research we present that it will be enough to properly identify previous file (Gladyshev, 2005).

2.2 Mathematical Model

We explain process of formation slack space with formal mathematical model based on Gladyshev (2005) work. The basic cluster model we describe can store data objects of only three possible lengths:

$$LENGTH = \{0,1,2\} \quad (1)$$

Where zero length means that the cluster is unallocated (It is not equal statement that there is no data stored on media. Either after formatting or wiping cluster out it can be filled with random data or byte patterns like "00". Actually we cannot prove if its not a part of valid file or data e.g. cryptographic container). Length 1 means that the cluster contains only one object of the size of the unrelated data, and the length 2 means that the cluster contains both object of the data block "x", end "y". All other sizes are disallowed in this model. This assumptions, divides the cluster into two parts which are shown in Figure 3; the left part that in the final state contains the unrelated data "u" and "y", and the right part that in the final state contains the piece of recovered data "x".

$$\begin{aligned} LEFT\_PART &= \{u,x1,y1\} \\ RIGHT\_PART &= \{u,y2,x2\} \end{aligned} \quad (2)$$

Advanced cluster model (ACM) we created is not limited to one cluster of data, we take whole disk space, where n is number of block series.

$$ACM = \{u, x_1, y_1\}\{u, x_2, y_2\} + \{u, x_3, y_3\}\{u, x_4, y_4\} + \{u, x_{(n-1)}, y_{(n-1)}\} \{u, x_{(m-1)}, y_{(m-1)}\} + \{u, x_n, y_n\}\{u, x_m, y_m\} \quad (3)$$

As we see as a final state we will have n parts of our evidence X file. In the best possibility, n can have size of evidence logical file, this is case where file was deleted but not overwritten. More often we will get less than 512 Byte part of X file in slack space, and other parts in an unallocated space. We take assumption that border of left and right part of each basic cell of the cluster model is the end of sector.

3 N-BYTE HASH

From a mathematical model we can see that standard hashing algorithms will not work when dealing with partially erased files. We cannot predict which part of the file we will be able to recover, that is main reason why reducing input data length to less than length of file is necessary. There is algorithm (Kornblum, 2006) which takes blocks as input $H(X_{p(1\text{-}512)})$, in most cases file system block has 512 Bytes but there are disadvantages of this option(Menezes, 1996). Blocks can have other values than 512 bytes depending on file system used, it's very hard to convert algorithm and correlated hash tables to work with file systems with different block bit length (Henson, 2003). The next disadvantage is that we can miss a part of evidence file in its last block because we cannot surely predict ram slack data entry. The Ram slack we can explain in mathematical model. in figure 1 Y file ends just in the end of 5 block in a cluster. More likely it would end in the middle of the block. In this case there will be ram slack space created to the end of the block (most of Operation Systems to deal with this problem, makes a wiping till end of the block, however in older MS systems it could be random data from Random Access Memory, this is actually why it's called RAM slack). The third disadvantage of using block input is performance. Taking 512 Bytes blocks force us to make hash for every byte on hard disk. Hashing every byte on disk is essential when we use hash function to preserve evidence, this is one of the most important item in creating chain of custody. However in our apply it is unnecessary and not efficient. Solution performance is depending on two main factors, the first is number of I/O operations on hard drive. Hard disk read/write operations, and interface for connecting drives is still bottleneck in computers. The second performance factor is computation time of hash function. Cryptographic hash functions are designed to be fast in both hardware and software implementation, but it is obvious that they have impact on computer performance. That is why we focus on n<512 versions of block hashing. In computer forensics there are widely used two cryptographic hash function MD5 (Ronald Rivest, Message-Digest algorithm 5) with a 128-bit hash value and SHA-1 designed by the National Security Agency (NSA) which creates a 160 bit message output based on principles similar to those used in MD5. In this research we will focus on Message Digest algorithm (White, 2005).

The MD5 cryptographic function algorithm first divides the data input into 512 bits blocks (Menezes, 1996). At the end of the last block 64 Bits are inserted to record the length of the original input. If input is smaller, bit value is filled with 0 to 448 bit block. Padding function is performed in the following way: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512.

This effects minimum input of N-Byte sector hashing considered 448 bits (56 bytes) for one full round of algorithm. And that is why we choose 56 bytes as input length in our algorithm. We considered also 120 Bytes input (two full round of md5). Standard full block input will have 512 Bytes of data + 64bits of length record + 448 bit padding, this results 576 Bytes MD5 input (9 full rounds).

Creating the hash tables compatible with presented algorithm should take into consideration that the same records can be ascribed to several different files. And that there will be several hash records to each file depending on its length. This characteristic is described wider in research implementation section.

4 PRACTICAL RESEARCH IMPLEMENTATION

We have implemented function $h(X_{p(1\ldots n)})$ based on Massage Digest 5 cryptographic hash function algorithm. We have performed several tests using the same software and hardware environment with n equal 48, 112 and 512. Tests were carried out to show efficiency of each method. Tests where repeated 20 times to determine and reduce error rate.

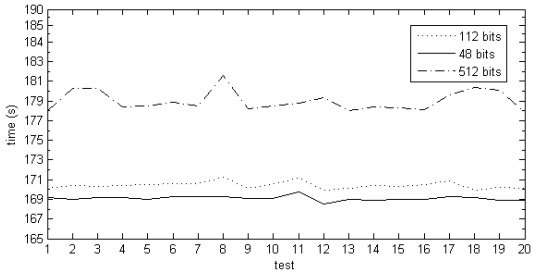

Figure 2: N-byte hash function comparison.

Test results confirm that process time is strictly correlated with input length of cryptographic hash function. Difference between $h\{X_{p(1...48)}\}$ and $h\{X_{p(1...112)}\}$ is minimal but clearly point on efficiency growth by reducing Message-Digest algorithm rounds amount. Comparing with $h\{X_{p(1...512)}\}$ method we gain more than 8% time reduction. Time reduction is especially useful during analysis of TB hard drives which become more and more popular.

When we use low inputs to decrease process time, we have to be aware of potential threats. The first is increase of positive negative hits. The hits are correlated with high possibility of collision occurrence we explained in theory section. On less random data set we should deal with higher amount of collisions because of regular shorter input values. High probability of collisions occurring is a reason we recommend using n equal 120 byte input data length. With relatively low increase of process time we get much more collision free function which will reduce necessity of manual check.

5 CONCLUSIONS

We have proposed an algorithm which is based on hash function which generates values based on N-Bytes input from each data sector. N-Byte slack space hashing can be used as a more efficient replacement of block file hashing for identifying partially recovered files during data retrieving process. In laboratory tests we obtain 8% time decrease using h{Xp(1...48)} or h{Xp(1...112)} hashing instead of full block 512 Byte input algorithm. In a real word environment it can accelerate computer data analysis. The efficiency of N-Byte slack space hashing results from construction and implementation of md5 cryptographic hash function which is widely used in the computer forensics. Further research will include the research on SHA (Secure Hash Algorithm) performance compared with use of md5 and additional tests in real environment. Also the research on identifying most occurring data inputs will be processed to create predefined n-bit inputs. It will be used to create tables similar to "rainbow tables" to increase efficiency. It should be stated simply that N-Byte Slack Space Hashing is not replacement of traditional hash analysis in computer forensics. Our solution is created to focus on a narrow problem of identifying partially recovered data. This is area in which standard hash analysis does not work properly.

REFERENCES

Kornblum, J., 2006. *Identifying almost identical files using context triggered piecewise hashing*, DFRWS Digital Investigation, Elsevier, 91 –97.

Stein, B., 2005. *Fuzzy-Fingerprints for Text-Based Information Retrieval*, Bauhaus University Weimar, Germany, Journal of Universal Computer Science, 572-579, ISSN 0948-695.

Bunting, S., 2008. *The Official EnCase Certified Examiner Study Guide*, Wiley Publishing, ISBN: 978-0-470-18145-4.

Gladyshev P., 2005. *Finite State Machine Analysis of a Blackmail Investigation*, International Journal of Digital Evidence, 4(1).

White, D., 2005. *NIST National Software Reference Library*. National Institute of Standards and Technology.

Menezes, A., 1996. *Handbook of Applied Cryptography*, CRC Press.

Henson, V., 2003. *An Analysis of Compare-by-hash*, Ninth Workshop on Hot Topics in Operating Systems HotOS-IX, Lihue, Hawaii, USA.

Microsoft, 2004. *Description of the FAT32*, ID310524, Microsoft.

Breeuwsma, M., 2007. *Forensic Data Recovery from Flash Memory*, Small Scale Digital Device Forensics Journal, 1(1).

Berghel, H,. 2007. *Hiding data, forensics, and anti-forensics*, Communications of the ACM.

Microsoft, 2000. *FAT: General Overview of On-Disk Format. FAT32 File System Specification, Version 1.03.*

Casey, E., 2004. *Tool Review—WinHex.* Journal of Digital Investigation, 1(2).