Chapter 8

IMPROVING DISK SECTOR INTEGRITY USING *K*-DIMENSION HASHING

Zoe Jiang, Lucas Hui and Siu-Ming Yiu

Abstract The integrity of data stored on a hard disk is typically verified by computing the chained hash value of disk sector data in a specific order. However, this technique fails when one or more sectors turn bad during storage, making it impossible to compute their hash values. This paper presents a k-dimension hashing scheme, which computes and stores multiple hash values for each hard disk sector. The hash values for each sector are computed in different ways; thus, when a hard disk develops bad sectors, it is still possible to verify the integrity of the data in the unaffected sectors. The paper also discusses how hashing parameters may be tuned to achieve desirable properties, including minimizing the probability that the integrity of a sector cannot be verified because other sectors have gone bad.

Keywords: Evidence integrity, hard disks, hash values, k-dimension hashing

1. Introduction

This paper focuses on a common, but important, problem in digital forensic investigations: Suppose certain data was written to a hard disk when it was created for evidentiary purposes; after a period of time – say one month – how could one prove that the hard disk contents are the same as before?

The straightforward scheme is to calculate a chained hash value of all the data in all the sectors in a specific sequence. This hash value is digitally signed and stored in a secure location. At some point in the future, when the integrity of the hard disk must be evaluated, the chained hash value is recomputed and compared with the previous value. If the two hash values match, the hard disk content is assumed not to

have been modified; if the values do not match, data in one or more disk sectors is somehow different from the original data.

The chained hashing scheme fails when the stored hard disk develops one or more bad sectors. A hash value cannot be computed for a bad sector and, consequently, the chained hash value for the entire hard disk cannot be calculated. Moreover, as disk capacity increases, the number of sectors increases, which makes the chained hashing scheme less attractive.

This paper describes an improved hashing scheme, which computes and stores multiple hash values for hard disk sectors. Specifically, hash values computed in different ways are available for verifying the integrity of a sector. Thus, when a hard disk develops one or more bad sectors, it is still possible to verify the integrity of the data in the unaffected sectors.

2. Background

This section describes the physical structure of hard disks and discusses hashing techniques for verifying the integrity of stored data.

2.1 Hard Disk Structure

A hard disk has one or more platters for storing data. Each platter has two read/write heads, one for the top face of the platter and the other for the bottom face. A platter is divided into tens of thousands of tightly-packed concentric circles called tracks. A cylinder is the set of tracks at which the heads are currently located.

Since tracks hold far too much information to be suitable as the smallest individually-addressable units of storage on a disk, each track is further divided into sectors that typically hold 512 bytes of data. Modern hard disks may have several thousand sectors in a single track.

An individual sector is traditionally addressed using an ordered CHS triple containing the cylinder, head and sector numbers (Figure 1). Due to the 8.4 GB limit of the Int 13h interface, modern drives are no longer specified using the CHS mode. Instead, they are addressed at the logical level using logical block addressing (LBA). At the physical level, however, most modern hard disks still use the CHS mode. Therefore, by accessing the integrated disk controller, which automatically translates LBA to the physical geometry, it is possible to match CHS triples to the physical hard disk characteristics [7].

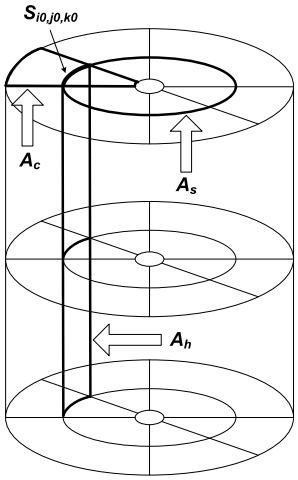

Figure 1. Hard disk structure.

2.2 Verifying Hard Disk Integrity

Most digital forensic tools (e.g, EnCase [3] and DESK [1]) use the chained hashing scheme to verify the integrity of data on hard disks. A change to just one bit in a sector, file or hard disk causes the hash value to be different [2].

Kornblum [5] recently proposed the context triggered piecewise hashing (CTPH) scheme to identify modified versions of known files (where data may have been inserted, modified or deleted). Although the CTPH scheme was designed for files, it can be applied to hard disks – a bad sector is considered to correspond to the portion of a file that has been modified. However, the CTPH scheme has high computational time requirements of $O(n \log n)$ where n is the size of the data being hashed. It is, therefore, not feasible to apply CTPH to large capacity hard disks (e.g., those exceeding 120 GB).

Jiang, *et al.* [4] have proposed a 3-dimension hashing scheme with better performance than the CTPH technique. This scheme computes multiple hash values to reduce the impact of bad sectors on disk integrity verification while requiring only a linear ($O(n)$) increase in computational time. The 3-dimension hashing scheme calculates hash values for: (i) all sectors with the same cylinder and head numbers (A_s in Figure 1) for all cylinder and head numbers; (ii) all sectors with the same cylinder and sector numbers (A_h in Figure 1) for all cylinder and sector numbers; and (iii) all sectors with the same head and sector numbers (A_c in Figure

1) for all head and sector numbers. Thus, every hash value that is stored has a physical meaning.

3. *k*-Dimension Hashing Scheme

The 3-dimension hashing scheme significantly reduces the probability that one or more bad sectors will affect the integrity verification of a hard disk. However, it has some limitations. A major drawback is that it is not always possible to obtain information about the physical structure of the hard disk; this is mainly due to the large capacities of modern hard disks and the diversity of technologies they employ [7]. For example, a USB thumb drive that uses solid state technology requires an integrity checking scheme that does not involve physical drive characteristics.

The k-dimension scheme described in this section extends 3-dimension hashing by using an arbitratry k ($k > 0$). This provides more freedom to design hashing schemes, including schemes that do not rely on the physical characteristics of hard disks. The only requirement is that the sectors in a hard disk being verified form a sequence.

Let N be the total number of disk sectors in a hard disk, and let p be the probability that any one disk sector becomes a bad sector after some period of time. We investigate the fail probability (P_f) of an integrity proof of a disk sector. This occurs when all the hash values involving the disk sector cannot be computed because other sectors involved in the hash computations have gone bad.

A 1-dimension hashing scheme is the trivial case that computes one hash value for all N sectors. The integrity proof of a disk sector is viable only when all the sectors are good sectors, which occurs with probability $(1-p)^{(N-1)}$. Consequently, the fail probability P_f is $1-(1-p)^{(N-1)}$.

For a 2-dimension scheme, the N sectors give rise to an $N_1 \times N_2$ (2-dimensional) array where N_1 and N_2 are integers such that $N_1 \times N_2 = N$. The minimum value of P_f occurs when $N_1 = N_2 = N^{1/2}$. In this case, the probability P_f is equal to $1 - \{(1-p)^{[N^{(\frac{1}{2})}-1]}\}^2$.

Similarly, for a k-dimension hashing scheme, where the sectors form a k-dimensional array, the minimum value of P_f occurs when the size of each dimension N_k is equal to $N^{1/k}$. Therefore, for a k-dimension scheme with $k \geq 1$, $P_f = \{1-(1-p)^{[N^{(\frac{1}{k})}-1]}\}^k$.

Note that extra hash values must be stored when implementing the k-dimension hashing scheme. In general, the total number of hash values (Num) stored is equal to $k \cdot N^{(\frac{k-1}{k})}$.

Increasing the number of dimensions k decreases the fail probability P_f, but the number of hash values Num also increases. It is, therefore, necessary to examine how P_f may be reduced while Num is also reduced.

One strategy is to divide the N disk sectors into j blocks ($j \geq 1$) and apply the k-dimension hashing scheme to each individual block. This strategy is simple and effective. Even in the 1-dimension case, by setting j to N, the probability P_f can be reduced to 0 with Num set equal to N! This is the absolute minimum value of P_f; therefore, it is necessary to consider the combined effect of the dimension size k and the number of blocks j.

Upon substituting the number of blocks j in place of the number of disk sectors N, the fail probability for the k-dimension hashing scheme is given by:

$$P_f = \{1 - (1-p)^{[(\frac{N}{j})^{(\frac{1}{k})} - 1]}\}^k. \tag{1}$$

The corresponding number of hash values to be stored is given by:

$$Num = j \cdot k \cdot (\frac{N}{j})^{(\frac{k-1}{k})}. \tag{2}$$

4. Analysis of k-Dimension Hashing

Tables 1–4 present the fail probabilities and the numbers of hash values required to be stored for various values of N (number of sectors) and p (probability that a sector becomes bad).

To simplify the presentation and related discussion, the data in Tables 1–4 is plotted to create the graphs in Figures 2 through 7. Figures 2 and 3 present the data in Table 1. Figures 2 and 4 present the data in Table 2. Figures 5 and 6 present the data in Table 3. Figures 5 and 7 present the data in Table 4. Note that Figures 2, 3 and 4 correspond to $N = 1.152e8$ while Figures 5, 6 and 7 correspond to $N = 3.6e8$.

As expected, increasing the number of dimensions k while keeping the number of blocks j fixed yields a lower fail probability P_f. However, the data also reveals that, when k is increased by 1, P_f drops by a value of approximately p. This anomaly can be partially explained by simplifying Equation 1 above. Given that $(1-e)^m$ can be approximated by $1 - em$ when e is very small and integer $m > 1$, the equation for P_f simplifies to:

$$\{p \cdot [(N/j)^{(1/k)} - 1]\}^k.$$

Upon further simplification and ignoring the -1 term, P_f is given by:

$$p^k \cdot (N/j).$$

Table 1. P_f and Num for $N = 1.152e8$, $p = 1e-5$.

j/k		**1-D**	**2-D**	**3-D**	**4-D**
$1e0$	P_f	1	$1.04e-2$	$1.10e-7$	$1.11e-12$
	Num	1	$2.15e4$	$7.10e5$	$4.45e6$
$1e1$	P_f	1	$1.10e-3$	$1.10e-8$	$1.07e-13$
	Num	$1.00e1$	$6.79e4$	$1.53e6$	$7.91e6$
$1e2$	P_f	1	$1.00e-4$	$1.10e-9$	$1.02e-14$
	Num	$1.00e2$	$2.15e5$	$3.30e6$	$1.41e7$
$1e3$	P_f	$6.84e-1$	$1.00e-5$	$1.10e-10$	$9.21e-16$
	Num	$1.00e3$	$6.79e5$	$7.10e6$	$2.50e7$
$1e4$	P_f	$1.09e-1$	$1.00e-6$	$1.00e-11$	$7.67e-17$
	Num	$1.00e4$	$2.15e6$	$1.53e7$	$4.45e7$
$1e5$	P_f	$1.14e-2$	$1.08e-7$	$8.50e-13$	$5.42e-18$
	Num	$1.00e5$	$6.79e6$	$3.30e7$	$7.91e7$
$1e6$	P_f	$1.14e-3$	$9.47e-9$	$5.80e-14$	$2.68e-19$
	Num	$1.00e6$	$2.15e7$	$7.10e7$	$1.41e8$
$1e7$	P_f	$1.05e-4$	$5.73e-10$	$2.00e-15$	$5.03e-21$
	Num	$1.00e7$	$6.73e7$	$1.50e8$	$2.50e8$
$1e8$	P_f	$1.52e-6$	$5.37e-13$	$1.10e-19$	$1.68e-26$
	Num	$1.00e8$	$2.15e8$	$3.30e8$	$4.45e8$
N	P_f	0	0	0	0
	Num	$1.15e8$	$2.30e8$	$3.50e8$	$4.61e8$

Table 2. P_f and Num for $N = 1.152e8$, $p = 1e-10$.

j/k		**1-D**	**2-D**	**3-D**	**4-D**
$1e0$	P_f	$1.15e-2$	$1.15e-12$	$1.14e-22$	$1.11e-32$
	Num	1	$2.14e4$	$7.10e5$	$4.45e6$
$1e1$	P_f	$1.15e-3$	$1.15e-13$	$1.14e-23$	$1.10e-33$
	Num	$1.00e1$	$6.79e4$	$1.53e6$	$7.91e6$
$1e2$	P_f	$1.15e-4$	$1.15e-14$	$1.12e-24$	$1.02e-34$
	Num	$1.00e2$	$2.15e5$	$3.30e6$	$1.41e7$
$1e3$	P_f	$1.15e-5$	$1.15e-15$	$1.08e-25$	$9.22e-36$
	Num	$1.00e3$	$6.79e5$	$7.10e6$	$2.50e7$
$1e4$	P_f	$1.15e-6$	$1.13e-16$	$1.01e-26$	$7.68e-37$
	Num	$1.00e4$	$2.15e6$	$1.53e7$	$4.45e7$
$1e5$	P_f	$1.15e-7$	$1.09e-17$	$8.53e-28$	$5.42e-38$
	Num	$1.00e5$	$6.79e5$	$3.30e7$	$7.91e7$
$1e6$	P_f	$1.14e-8$	$9.47e-19$	$5.78e-29$	$2.68e-39$
	Num	$1.00e6$	$2.15e7$	$7.10e7$	$1.41e8$
$1e7$	P_f	$1.05e9$	$5.73e-20$	$1.99e-30$	$5.03e-41$
	Num	$1.00e7$	$6.79e7$	$1.53e8$	$2.50e8$
$1e8$	P_f	$1.52e-11$	$5.37e-23$	$1.13e-34$	$1.68e-46$
	Num	$1.00e8$	$2.45e8$	$3.30e8$	$4.45e8$
N	P_f	0	0	0	0
	Num	$1.15e8$	$2.30e8$	$3.50e8$	$4.61e8$

Table 3. P_f and Num for $N = 3.6e8$, $p = 1e-10$.

j/k		**1-D**	**2-D**	**3-D**	**4-D**
$1e0$	P_f	1	$3.00e-2$	$3.50e-7$	$3.50e{-}12$
	Num	1	$3.79e4$	$1.52e6$	$1.05e7$
$1e1$	P_f	1	$3.40e-3$	$3.50e-8$	$3.41e{-}13$
	Num	$1.00e1$	$1.20e5$	$3.27e6$	$1.86e7$
$1e2$	P_f	1	$3.53e-4$	$3.50e-9$	$3.28e{-}14$
	Num	$1.00e2$	$3.79e5$	$7.04e6$	$3.30e7$
$1e3$	P_f	$9.73e-1$	$3.56e-5$	$3.40e{-}10$	$3.05e{-}15$
	Num	$1.00e3$	$1.20e6$	$1.50e7$	$5.88e8$
$1e4$	P_f	$3.02e-1$	$3.56e-6$	$3.30e{-}11$	$2.66e{-}16$
	Num	$1.00e4$	$3.79e6$	$3.30e7$	$1.05e8$
$1e5$	P_f	$3.50e-2$	$3.48e-7$	$2.90e{-}12$	$2.07e{-}17$
	Num	$1.00e5$	$1.20e7$	$7.00e7$	$1.86e8$
$1e6$	P_f	$3.60e-3$	$3.23e-8$	$2.30e{-}13$	$1.27e{-}18$
	Num	$1.00e6$	$8.79e7$	$1.50e8$	$3.32e8$
$1e7$	P_f	$3.00e-4$	$2.50e-9$	$1.20e{-}14$	$4.41e{-}20$
	Num	$1.00e7$	$1.20e8$	$3.30e8$	$5.88e8$
$1e8$	P_f	$3.00e-5$	$8.05e{-}11$	$1.50e{-}16$	$2.03e{-}22$
	Num	$1.00e8$	$3.79e8$	$7.99e8$	$1.05e9$
N	P_f	0	0	0	0
	Num	$4.00e8$	$7.20e8$	$1.10e9$	$1.44e9$

Table 4. P_f and Num for $N = 3.6e8$, $p = 1e-10$.

j/k		**1-D**	**2-D**	**3-D**	**4-D**
$1e0$	P_f	$3.54e-2$	$3.60e{-}12$	$3.60e{-}22$	$3.50e{-}32$
	Num	1	$3.79e4$	$1.52e6$	$1.05e7$
$1e1$	P_f	$3.60e-3$	$3.60e-1$	$3.60e{-}23$	$3.42e{-}33$
	Num	$1.00e1$	$1.20e5$	$3.27e6$	$1.86e7$
$1e2$	P_f	$4.00e04$	$3.60e{-}14$	$3.50e{-}24$	$3.28e{-}34$
	Num	$1.00e2$	$3.79e5$	$7.04e6$	$3.30e7$
$1e3$	P_f	$4.00e-5$	$3.59e{-}15$	$3.50e{-}25$	$3.05e{-}35$
	Num	$1.00e3$	$1.20e6$	$1.50e7$	$5.88e8$
$1e4$	P_f	$4.00e-6$	$3.56e{-}16$	$3.30e{-}26$	$2.66e{-}36$
	Num	$1.00e4$	$3.79e6$	$3.30e7$	$1.05e8$
$1e5$	P_f	$4.00e-7$	$3.48e{-}17$	$2.90e{-}27$	$2.07e{-}37$
	Num	$1.00e5$	$1.20e7$	$7.00e7$	$1.86e8$
$1e6$	P_f	$4.00e-8$	$3.23e{-}18$	$2.30e{-}28$	$1.27e{-}38$
	Num	$1.00e6$	$8.79e7$	$1.50e8$	$3.32e8$
$1e7$	P_f	$4.00e-9$	$2.50e{-}19$	$1.20e{-}29$	$4.41e{-}40$
	Num	$1.00e7$	$1.20e8$	$3.30e8$	$5.88e8$
$1e8$	P_f	$3.00e{-}10$	$8.05e{-}21$	$1.50e{-}31$	$2.03e{-}42$
	Num	$1.00e8$	$3.79e8$	$7.99e8$	$1.05e9$
N	P_f	0	0	0	0
	Num	$4.00e8$	$7.20e8$	$1.10e9$	$1.44e9$

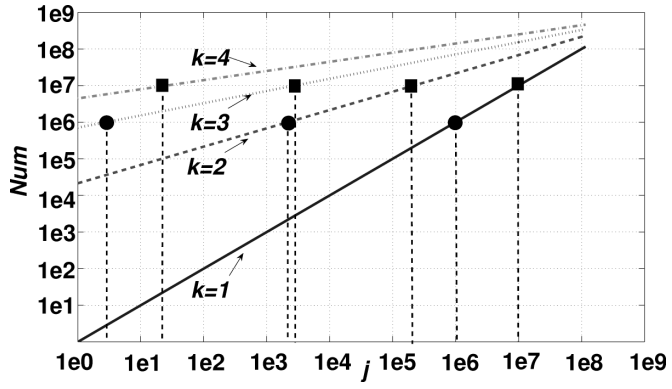

Figure 2. *Num* versus j for $N = 1.152e8$.

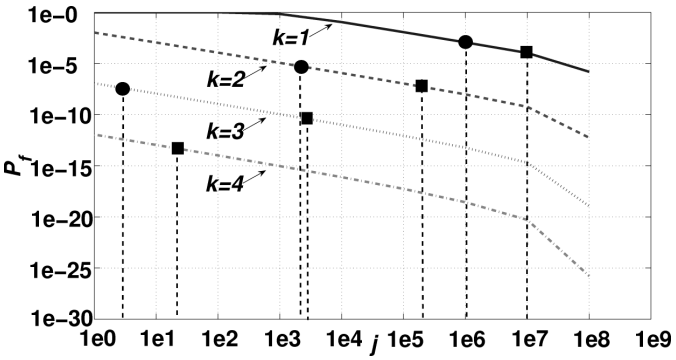

Figure 3. P_f versus j for $N = 1.152e8$, $p = 1e-5$.

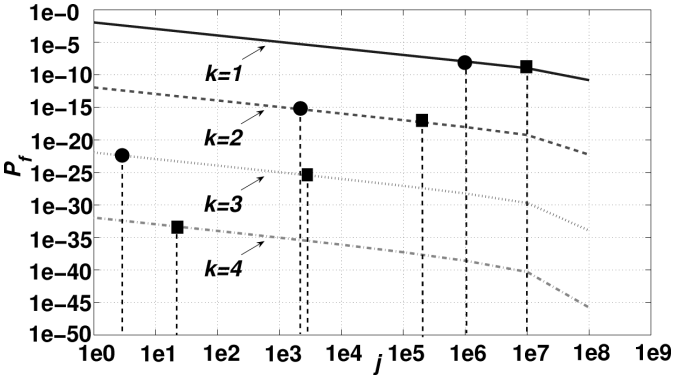

Figure 4. P_f versus j for $N = 1.152e8$, $p = 1e-10$.

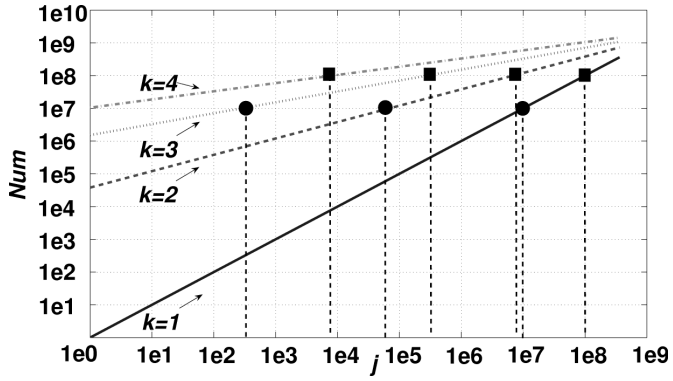

Figure 5. *Num* versus j for $N = 3.6e8$.

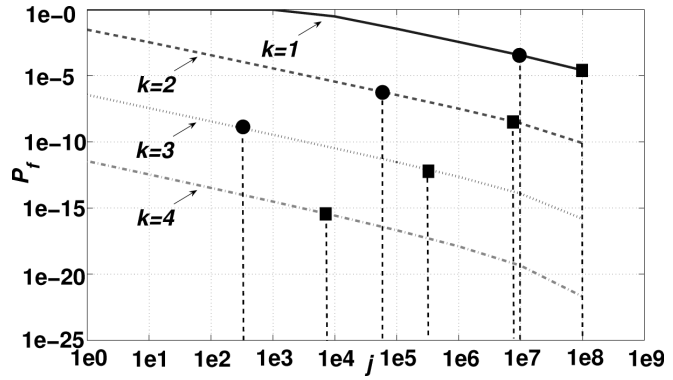

Figure 6. P_f versus j for $N = 3.6e8$, $p = 1e - 5$.

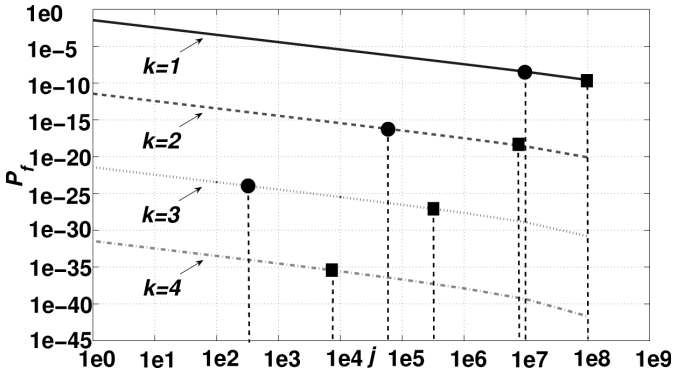

Figure 7. P_f versus j for $N = 3.6e8$, $p = 1e - 10$.

Therefore, for fixed N and j, every increment in k reduces P_f by a factor of p.

This leads to the observation that even if j is changed (but not by too much), it is beneficial to use a higher dimension to reduce the probability P_f. The reduction in P_f due to higher k is parameterized by p. Specifically, it is advantageous to use a higher dimension when probability p is low.

Note that in practice the expected number of bad sectors in a hard disk is low and the probability p is very low. For example, when $N = 1.152e8$ and $p = 1e-5$, the expected number of bad sectors is more than 1,000, which is not realistic. Our studies indicate that $p = 1e-10$ is a more realistic value. Nevertheless, the data corresponding to $p = 10e-5$ is presented to show the behavior of the hashing scheme for a p value that is not very small.

Another observation from the graphs is that although P_f is expected to drop to zero for $j = N$, this does not occur even when j is close to N (see Figures 4 and 7).

Figures 2–4 can be used to determine the appropriate number of dimensions to be used given a fixed *Num* (number of hash values to be stored). First, Figure 2 is used to determine the number of blocks (j) for each dimension value (k) that will require *Num* hash values. Next, Figure 3 or 4 is used to determine the probabilities P_f corresponding to the j values for each value of k. Finally, the value of k that yields the lowest fail probability P_f is selected.

To illustrate the methodology, consider a fixed *Num* value of $1e7$. The four squares in Figure 2 identify the points with this *Num* value and k = 4, 3, 2 and 1. The j values of these four points are recorded. Next, the four points in Figure 3 with these j values and k = 4, 3, 2, 1 are identified (these are marked as squares in Figure 3). The P_f values corresponding to these four points can then be read from Figure 3. The lowest fail probability P_f occurs for $k = 4$. Similar analysis can be performed using Figures 5–7.

Upon investigating several different *Num* values, we have observed that it is better to use a higher dimension value k provided that *Num* is at least the minimum number of hash values needed by dimension k. Two examples in Figures 2–4 and Figures 5–7 illustrate the effect of increasing the dimension. The squares and circles in the figures correspond to *Num* values of $1e7$ and $1e6$, respectively. In both cases, it is clear that for the given *Num* value, a higher dimension value k yields a lower fail probability P_f. Upon comparing the two groups of points (squares and circles), it is apparent that a higher j value produces a lower fail probability P_f for the same dimension k.

5. Observations

Our analysis indicates that k-dimension hashing is very effective at reducing the fail probability P_f. For example, the fail probability for 10 blocks (with $p = 1e-10$ and $N = 1.152e8$) reduces from $1.15e-2$ in the straightforward scheme of using one hash value for the entire hard disk to $1.10e-33$ when 4-dimension hashing is used. This is a drastic decrease in fail probability. Similar reductions occur for other parameter settings.

Our findings can be summarized in the following recommendations. If the minimization of the fail probability P_f is the principal goal and Num hash values can be stored, where $Num < N$ (number of disk sectors), then it is best to use the highest possible k-dimension hashing scheme. On the other hand, if Num is close to or larger than N, then the 1-dimension hashing scheme with $P_f = 0$ is the best choice.

Note that these recommendations ignore the overhead involved in handling large numbers of hash values, especially when the hash values have to be digitally signed (as in many digital forensic tools [1, 3]). The Merkle hash tree [6] is a low overhead approach for signing multiple hash values [8]. Nevertheless, it is important to investigate the effect of the overhead involved in digital signing on the choice of dimension.

6. Conclusions

The k-dimension hashing scheme is a robust technique for verifying the integrity of data stored on hard disks. The scheme computes the hash values for each sector in multiple ways; thus, when one or more sectors go bad, it is still possible to verify the integrity of the data in the unaffected sectors. Our future research will investigate applications of k-dimension hashing to enhancing evidence preservation and detecting evidence tampering with high probability.

Acknowledgements

This research was partially supported by the Research Grants Council of the Hong Kong Special Administrative Region under Project Nos. HKU 7136/04E and HKU 7132/06E.

References

[1] K. Chow, C. Chong, K. Lai, L. Hui, K. Pun, W. Tsang and H. Chan, Digital evidence search kit, *Proceedings of the First International Workshop on Systematic Approaches to Digital Forensic Engineering*, pp. 187–194, 2005.

[2] J. Foster and V. Liu, Catch me, if you can, presented at *Black Hat Japan 2005* (www.blackhat.com/presentations/bh-usa-05/bh-us-05-foster-liu-update.pdf), 2005.

[3] Guidance Software, EnCase, Pasadena, California (www.guidance software.com).

[4] Z. Jiang, L. Hui, K. Chow, S. Yiu and P. Lai, Improving disk sector integrity using a 3-dimension hashing scheme, *Future Generation Communication and Networking*, vol. 2, pp. 141–145, 2007.

[5] J. Kornblum, Identifying almost identical files using context triggered piecewise hashing, *Proceedings of the Sixth Digital Forensic Research Workshop*, 2006.

[6] R. Merkle, A certified digital signature, *Proceedings of the Ninth International Cryptology Conference*, pp. 218–238, 1989.

[7] The PC Guide, Hard Disk Drives (www.pcguide.com/ref/hdd).

[8] M. Wang, S. Yiu, L. Hui, C. Chong, K. Chow, W. Tsang, H. Chan and K. Pun, A hybrid approach for authenticating MPEG-2 streaming data, *Proceedings of the International Workshop on Multimedia Content Analysis and Mining*, pp. 203–212, 2007.