Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

k-Dimensional hashing scheme for hard disk integrity verification in computer forensics*

Zoe Lin JIANG[†§1,2], Jun-bin FANG[†‡§2], Lucas Chi Kwong HUI[2], Siu Ming YIU[2], Kam Pui CHOW[2], Meng-meng SHENG[2]

([1]*School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen 518055, China*)

([2]*Department of Computer Science, The University of Hong Kong, Hong Kong, China*)

[†]E-mail: {zoeljiang, junbinfang}@gmail.com

Received Dec. 11, 2010; Revision accepted Mar. 29, 2011; Crosschecked Sept. 1, 2011

Abstract: Verifying the integrity of a hard disk is an important concern in computer forensics, as the law enforcement party needs to confirm that the data inside the hard disk have not been modified during the investigation. A typical approach is to compute a single chained hash value of all sectors in a specific order. However, this technique loses the integrity of all other sectors even if only one of the sectors becomes a bad sector occasionally or is modified intentionally. In this paper we propose a k-dimensional hashing scheme, kD for short, to distribute sectors into a kD space, and to calculate multiple hash values for sectors in k dimensions as integrity evidence. Since the integrity of the sectors can be verified depending on any hash value calculated using the sectors, the probability to verify the integrity of unchanged sectors can be high even with bad/modified sectors in the hard disk. We show how to efficiently implement this kD hashing scheme such that the storage of hash values can be reduced while increasing the chance of an unaffected sector to be verified successfully. Experimental results of a 3D scheme show that both the time for computing the hash values and the storage for the hash values are reasonable.

Key words: Computer forensics, Digital evidence, Hard disk integrity, k-Dimensional hashing
doi:10.1631/jzus.C1000425 **Document code:** A **CLC number:** TP309

1 Introduction

With the rapid development of Internet technology (Comito *et al.*, 2007) and electronic commerce (Hussain *et al.*, 2010), digital forensics has become more and more important in a perpetual race with criminals in the application of digital technologies. Nowadays, it is very common to have evidence existing in digital forms such as a deleted file in a hard disk of a suspect's computer. Due to the nature of digital information, which is easy to change, one of the biggest problems is how to effectively verify the integrity of evidence from the hard disk. Consider the following procedure: When a suspicious hard disk is seized by the police, it is investigated by the forensic expert to search for relevant evidence. Since the investigation process is always lengthy, the suspect (the hard disk owner) may challenge the validity of the evidence collected by the police. To argue that the evidence in the hard disk is valid (or invalid), a formal procedure is needed to provide integrity proof of the evidence, even after a long period. Therefore, an effective integrity verification method for a hard disk is required for both parties.

A simple solution, used in many digital forensics tools (e.g., EnCase (Garber, 2001) and DESK (Chow

‡ Corresponding author
§ The two authors contributed equally to this work
* Project supported by the Research Grants Council of Hong Kong SAR, China (No. RGC GRF HKU 713009E), the NSFC/RGC Joint Research Scheme (No. N_HKU 722/09), and HKU Seed Fundings for Basic Research (Nos. 200811159155 and 200911159149)

et al., 2005)), is to calculate one single chained one-way hash value for all sectors in a hard disk and store it securely for later verification. However, this method suffers from the risk that even one-bit change will collapse the integrity of the whole hard disk; i.e., it can provide only a yes-or-no answer about the integrity of the whole hard disk and is not accurate enough for the sectors. In some cases, we need more information beyond the yes-or-no answer, especially when the answer is NO. It may be necessary to know how many sectors are modified and where they are. In fact, even if the sectors are not modified intentionally, some of the sectors may become bad sectors (Schroeder and Gibson, 2007) during the investigation period (say, a few months to several years later). In addition, under some circumstances, the suspect is allowed to modify or even delete some sectors on his/her own, e.g., files classified as legal professional privilege data (Law *et al.*, 2008). If this occurs, the new hash value calculated will not equal the stored one. Therefore, it is desired to design a scheme that can identify which sectors affect the integrity, such that the integrity of the evidence on the hard disk can still be verified if the changed sectors are not related to the evidence.

The other extreme solution is to compute a hash value for each sector, and then sign and store all these signed hash values for later comparison. This approach is quite straightforward, but it needs to store too many hash values. For a hard disk with a capacity of 250 GB (488 392 065 sectors), it needs to store 488 392 065 hash values and requires about 7 GB storage if each hash value has 128 bits. Furthermore, if a wicked computer forensics expert modifies the content of one disk sector, he/she needs only to modify one hash value to fake the integrity proof. This leaves a vulnerability in the digital investigation tools.

In this paper, we propose a hashing scheme, called the kD scheme, which is a balance between the above two solutions. Roughly speaking, we use multiple hash chains and each sector is assigned to more than one chain so as to increase the chance of being verified successfully even if there are bad/modified sectors. Depending on the number of chains used, the number of hash values to be stored can be a lot smaller than that of the approach of using one hash value for every sector. Particularly, if k is designed as 1 or N (N is the number of sectors), the two special cases, 1D and ND schemes, are exactly the two solutions we described above.

Our scheme mainly includes three algorithms: the sector distribution algorithm that allocates sectors into chains so that each chain has more or less the same length; the one-pass algorithm that reads the whole hard disk once (to avoid extensive IO) and computes all required hash values; and the integrity check algorithm that verifies the integrity of a certain sector or a whole hard disk. We then give, in theoretical terms, the analysis and evaluation of the general kD scheme by tuning different parameters to meet various applications in practice. The storage of hash values and the probability of a sector that we may wrongly judge the integrity are two important criteria to measure the scheme, and these can be affected by the capacity of the hard disk, the probability for a sector being bad after some time, etc. Finally, we implement the scheme for $k = 3$ and evaluate our scheme using two hard disks of 60 GB and 250 GB, respectively.

2 Related work

In computer forensic area, examiners often need to understand and analyze a large amount of data that seem arbitrary to them. Cryptographic hash functions are often used by forensic examiners for data integrity checks. The National Software Reference Library (NSRL) was provided to identify known files by comparing several kinds of hash algorithms, based on two fundamental properties—collision resistance and being a one-way function (NIST, 2004). The first, being collision resistant, means that two different messages should not hash to the same value. The second property that good hash algorithms have is that they are pre-image resistant; i.e., it is computationally infeasible for a message to be constructed that matches a given hash. As a result, both MD5 and SHA-1 passed the examination as the cryptographic hash algorithms. Although MD5 and SHA-1 hash functions have been challenged in terms of their security due to collision attacks (Gauravaram *et al.*, 2006), the National Institute of Standards and Technology (NIST) also plans to add additional file signatures generated by other hash algorithms in the future, including those identified in FIPS (Federal Information Processing Standards) PUB 180-2 (SHA-256, SHA-384, SHA-512) (Mead, 2006).

Currently, there are several existing digital forensics tools (Garber, 2001; Chow *et al.*, 2005), among which EnCase is the most popular in computer forensic investigation. It provides intuitive graphical user interface (GUI), enhanced email/Internet support, and a powerful scripting engine. DESK is another tool which focuses more on Chinese language encoding. Both systems, together with many others, use the straightforward approach of one single chained hash value to verify the integrity of a hard disk. All existing tools face the same challenge that even a one-bit change to a data item (such as a known file or a disk) occurs, the integrity of the whole hard disk cannot be verified.

Harbour (2002) designed the piecewise hashing scheme which uses an arbitrary hashing algorithm to create many checksums, instead of just one, for a file. It was developed to mitigate errors during forensic imaging because an error can affect only one of the piecewise hashes. The remainder of the piecewise hashes can still be used to check the integrity of the remainder of the data. Kornblum (2006) improved the piecewise hashing scheme, called context triggered piecewise hash (CTPH), to identify known files that have had data inserted, modified, or deleted, in combination with rolling hash. Although they are originally designed for files, they can also be adopted to a hard disk (for a sector becoming bad can be considered as a part of a file being modified). However, they are not efficient enough to be used in real applications. For example, the CTPH scheme requires an $O(n \log n)$ running time where n is the data size, which is a very huge overhead when it is applied to a hard disk with a reasonable size such as 250 GB.

Jiang *et al.* (2007) proposed a cylinder-head-sector hashing scheme, $\mathcal{CHS}$ for short, to compute multiple hash values based on the mechanical structure of a hard disk. However, due to the large capacities of modern hard disks and the diversity of technologies employed (Chen *et al.*, 2006), it is not always possible to obtain information about the physical structure of the hard disk. For example, a USB thumb drive that uses solid state technology requires an integrity checking scheme that does not involve physical drive characteristics.

3 k-Dimensional hashing scheme

In this section, we describe how to design the kD scheme in detail. For a better understanding and visualization, we use a specific case of $k = 3$ as an example to illustrate the concept of our scheme.

3.1 Sector distribution algorithm

First, we design a sector distribution algorithm that sequentially maps all the sectors into a kD hypercube to help our scheme achieve the best performance. Assume that the hard disk has N sectors denoted as $S = \{s_j | j = 0, 1, \cdots, N-1\}$. Our goal is to rearrange the sectors into a 3D structure ($k = 3$ here), such that each sector is uniquely mapped to an ordered 3D array (d_3, d_2, d_1) in $\mathcal{D}_3$, $\mathcal{D}_2$, and $\mathcal{D}_1$ directions respectively, denoted as $s_{(d_3,d_2,d_1)}$.

The mapping from j to (d_3, d_2, d_1) is just like a mapping from a 1D chain to many 3D nested 'cubes' with the same coordinate origin s_0. As a result, the first sector s_0 is located in $(0, 0, 0)$; the first eight sectors $s_0, s_1, \cdots, s_7$ are located in (d_3, d_2, d_1) where $d_3, d_2, d_1 \in \{0, 1\}$; the first 27 sectors $s_0, s_1, \cdots, s_{26}$ are located in (d_3, d_2, d_1) where $d_3, d_2, d_1 \in \{0, 1, 2\}$; $\cdots$; the first h^3 sectors $s_0, s_1, \cdots, s_{h^3-1}$ are located in (d_3, d_2, d_1) where $d_3, d_2, d_1 \in \{0, 1, \cdots, h-1\}$. Note that when the number of sectors is not exactly h^3 for some integer h, we will let the outmost cube be partially filled and let all inner cubes be completely filled. Let $L = \lfloor j^{1/3} \rfloor$. We call L the 'layer number' of sector s_j. As a special case, s_0 is the smallest cube with $L = 0$, and is mapped directly into $s_{(0,0,0)}$. Since the first L^3 sectors (i.e., sectors $s_0, s_1, \cdots, s_{L^3-1}$) have been distributed in L nested cubes (with layer numbers 0 to $L-1$), the sector distribution algorithm will place sector s_j in the $(L+1)$th cube with layer number L.

Denote the mapping algorithm from the sector index j to the 3D coordinates (d_3, d_2, d_1) as $\mathcal{SDA}_3$. Obviously $\mathcal{SDA}_3(0) = (0, 0, 0)$ since the sector s_0 is also denoted as $s_{(0,0,0)}$. Now we want to find the exact mapping of $\mathcal{SDA}_3$.

First of all, consider the difference between the Lth cube and the $(L+1)$th cube. There is a set of sectors S that can be added outside the Lth cube to become the $(L+1)$th cube. We divide this set of points S into three extension planes with different priorities. The plane with a higher priority should be filled up earlier with sectors. Let $P_1(L)$ represent the first extension plane of layer L in $\mathcal{D}_1$ direction with the highest priority 3, which includes L^2 sectors. Detailedly, sectors in this plane are represented as $s_{(d_3,d_2,d_1)}$, where d_3 is from 0 to $L-1$, d_2 is from 0

to $L-1$, and d_1 is fixed to L. Similarly, the second extension plane $P_2(L)$ of layer L in $\mathcal{D}_2$ direction with priority 2, contains $L(L+1)$ sectors represented as $s_{(d_3,d_2,d_1)}$, where d_3 is from 0 to $L-1$, d_2 is fixed to L, and d_1 is from 0 to L. Finally, the third extension plane $P_3(L)$ of layer L in $\mathcal{D}_3$ direction with the lowest priority 1, contains $(L+1)^2$ sectors represented as $s_{(d_3,d_2,d_1)}$, where d_3 is fixed to L, d_2 is from 0 to L, and d_1 is from 0 to L.

After defining the three extension planes with different priorities to be filled up, we need to design how the sectors should be distributed in each of the planes as follows: with increasing j, sector s_j is mapped into $s_{(d_3,d_2,d_1)}$ in $P_1(L)$ satisfying that the pair (d_3,d_2) is such a nested loop that in the inner loop, d_2 increases from 0 to $L-1$, and in the outer loop, d_3 increases from 0 to $L-1$, with fixed $d_1=L$. Similarly, in $P_2(L)$, (d_3,d_2) varies with increasing j satisfying that d_1 increases from 0 to L in the inner loop and d_3 increases from 0 to $L-1$ in the outer loop, with fixed $d_2=L$. And in $P_3(L)$, (d_2,d_1) varies with increasing j satisfying that d_1 increases from 0 to L in the inner loop and d_2 increases from 0 to L in the outer loop, with fixed $d_3=L$.

After designing the sector distribution policy in each extension plane, the last step is to locate the plane in which the sector $s_j \in S$ will be placed and where its exact position is, according to the value of j. For convenience, i is defined as the index of s_j in the extension plane it locates.

If $L^3 < j+1 \le L^3+L^2$, s_j should be the $(i+1)$th sector in $P_1(L)$ according to the nested loop with $i=j-L^3$, as is the case described in Line 6 in Algorithm 1.

If $L^3+L^2 < j+1 \le L^3+L^2+L(L+1)$, s_j should be the $(i+1)$th sector in $P_2(L)$ according to the nested loop with $i=j-L^3-L^2$, as is the case described in Line 9 in Algorithm 1.

If $L^3+L^2+L(L+1) < j+1 \le (L+1)^3$, s_j should be the $(i+1)$th sector in $P_3(L)$ according to the nested loop with $i=j-L^3-L^2-L(L+1)$, as is the case described in Line 12 in Algorithm 1.

Formally, given a sector s_j, the mapping algorithm $\mathcal{SDA}_3(j)$ can be defined as Algorithm 1.

Example 1 Fig. 1 is a simple example for $\mathcal{SDA}_3(j)$. For $L=\lfloor j^{1/3} \rfloor = 2$, suppose the 0th and 1st cubes have been filled up by the first $L^3=8$ sectors. We have three cases depending on the value of j as follows: If $2^3 < j+1 \le 2^3+2^2$, such as $j=9$, let $i=j-2^3=1$ where i starts from 0. Then s_9 is the $(i+1=2)$nd sector in $P_1(2)$. Since pair (d_3,d_2) starts from $(0,0)$ and increases according to the nested loop, the second pair should be $(0,1)$. And s_9 is mapped into $s_{(0,1,2)}$. If $2^3+2^2 < j+1 \le 2^3+2^2+2(2+1)$, such as $j=15$, let $i=j-2^3-2^2=3$. Then s_{15} is the $(i+1=4)$th sector in $P_2(2)$. Accounting pair (d_3,d_1) from $(0,0)$, we obtain the 4th pair $(1,0)$ from the former three pairs $(0,0)$, $(0,1)$, and $(0,2)$. And s_{15} is mapped into $s_{(1,2,0)}$. Similarly, if $2^3+2^2+2(2+1) < j+1 \le 3^3$, such as $j=25$, let $i=j-2^3-2^2-2(2+1)=7$. s_{25} should be the $(i+1=8)$th sector in $P_3(2)$ and is mapped into $s_{(2,2,1)}$.

Algorithm 1: $\mathcal{SDA}_3$, sector distribution algorithm for 3D

```
   input  : j
   output: (d3, d2, d1)
 1 if j=0 then
 2     (d3, d2, d1) = (0, 0, 0);
 3     Exit;
 4 end
 5 L = ⌊j^(1/3)⌋;
 6 if L^3 < j + 1 ≤ L^3 + L^2 then
 7     i = j − L^3;
 8     (d3, d2, d1) = (i/L, i mod L, L);
 9 else if L^3 + L^2 < j + 1 ≤ L^3 + L^2 + L(L + 1)
   then
10     i = j − L^3 − L^2;
11     (d3, d2, d1) = (i/(L + 1), L, i mod (L + 1));
12 else if L^3 + L^2 + L(L + 1) < j + 1 ≤ (L + 1)^3
   then
13     i = j − L^3 − L^2 − L(L + 1);
14     (d3, d2, d1) = (L, i/(L + 1), i mod (L + 1));
15 end
```

The inverse function of $\mathcal{SDA}_3$, defined as $\mathcal{SDA}_3^{-1}(d_3,d_2,d_1)$ in Algorithm 2, by mapping (d_3,d_2,d_1) to j, is also useful when one knows the 3D structure of a certain sector and wants to compute its 1D value.

It is straightforward that $(d_3,d_2,d_1) \leftrightarrow j$ is a one-to-one mapping. Thus, we have the following theorem:

Theorem 1 Function $\mathcal{SDA}_3$ is a function from a set $X=\{j \mid 0 \le j \le N-1\}$ to a set $Y=\{(d_3,d_2,d_1) \mid 0 \le d_3 \le R_3, 0 \le d_2 \le R_2, 0 \le d_1 \le R_1\}$, where R_1, R_2, R_3 are decided by N, with the property that, for every (d_3,d_2,d_1) in Y, there is

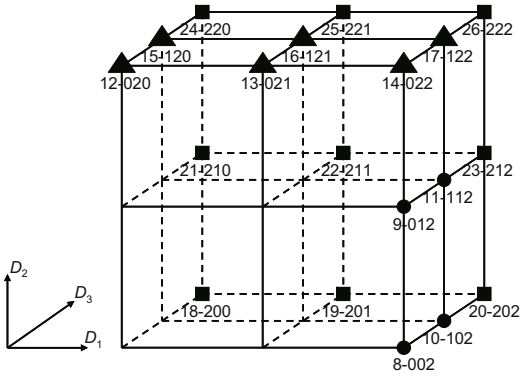

Fig. 1 An example for 3D sector distribution with $L = 2$. Solid circles, triangles, and rectangles represent sectors in $P_1(1)$, $P_2(1)$, and $P_3(1)$, respectively

Algorithm 2: $\mathcal{SDA}_3^{-1}$, inverse function of the sector distribution algorithm for 3D

input : (d_3, d_2, d_1)
output: j

if $(d_1 = 0, d_2 = 0, d_3 = 0)$ **then**
  $j = 0$;
  Exit;
end
$L = \max\{d_3, d_2, d_1\}$;
if $(d_1 > d_2)$ and $(d_1 > d_3)$ **then**
  $i = d_3 L + d_2$;
  $j = L^3 + i$;
else if $(d_2 \geq d_1)$ and $(d_2 > d_3)$ **then**
  $i = d_3(L + 1) + d_1$;
  $j = L^3 + L^2 + i$;
else if $(d_3 \geq d_2)$ and $(d_3 \geq d_1)$ **then**
  $i = d_2(L + 1) + d_1$;
  $j = L^3 + L^2 + L(L + 1) + i$;
end
.

exactly one j in X such that $\mathcal{SDA}_3(j) = (d_3, d_2, d_1)$. Or $\mathcal{SDA}_3$ is a bijection.

3.2 One-pass algorithm

Besides the sector distribution algorithm, implementing an efficient kD scheme also requires a fast hash calculation algorithm for the whole hard disk. In this subsection, the one-pass algorithm is presented to calculate all hash values by scanning the whole hard disk only once, which can greatly save time.

After running $\mathcal{SDA}_3(j)$ N times where $j = 0, 1, \cdots, N-1$ and based on Theorem 1, we obtain N triples $\{(d_3, d_2, d_1) \mid 0 \leq d_3 \leq R_3, 0 \leq d_2 \leq R_2, 0 \leq d_1 \leq R_1\}$, where R_1, R_2, R_3 depend on N. Let $v\_\mathcal{D}_1[d_3, d_2]$ be the hash value of a hash chain $c_{\mathcal{D}_1}[d_3, d_2]$ from the starting point of the chain, say, $s_{(d_3,d_2,0)}$, to the ending point, $s_{(d_3,d_2,R_1)}$, in the first dimension $\mathcal{D}_1$. Let $v\_\mathcal{D}_2[d_3, d_1]$ be defined similarly for $c_{\mathcal{D}_1}[d_3, d_1]$ from $s_{(d_3,0,d_1)}$ to $s_{(d_3,R_2,d_1)}$ in $\mathcal{D}_2$. Let $v\_\mathcal{D}_3[d_2, d_1]$ be defined similarly for $c_{\mathcal{D}_1}[d_2, d_1]$ from $s_{(0,d_2,d_1)}$ to $s_{(R_3,d_2,d_1)}$ in $\mathcal{D}_3$. $f(\cdot \parallel \cdot)$ can be a hash function provided in NIST (2004) and Mead (2006). We denote the algorithm as $\mathcal{OPA}_3$, shown in Algorithm 3.

Algorithm 3: $\mathcal{OPA}_3$, one-pass algorithm for 3D

input : $\{s_j \mid j = 0, 1, \cdots, N-1\}$
output: $\{v\_\mathcal{D}_1[d_3, d_2], v\_\mathcal{D}_2[d_3, d_1], v\_\mathcal{D}_3[d_2, d_1] \mid 0 \leq d_3 \leq R_3, 0 \leq d_2 \leq R_2, 0 \leq d_1 \leq R_1\}$

Set all $v\_\mathcal{D}_1[\,]$, $v\_\mathcal{D}_2[\,]$, and $v\_\mathcal{D}_3[\,]$ to the initial values;
foreach sector s_j where $j = 0, 1, \cdots, N-1$ **do**
  $(d_3, d_2, d_1) = \mathcal{SDA}_3(j)$;
  $v\_\mathcal{D}_1[d_3, d_2] = f(v\_\mathcal{D}_1[d_3, d_2] \parallel s_{(d_3,d_2,d_1)})$;
  $v\_\mathcal{D}_2[d_3, d_1] = f(v\_\mathcal{D}_2[d_3, d_1] \parallel s_{(d_3,d_2,d_1)})$;
  $v\_\mathcal{D}_3[d_2, d_1] = f(v\_\mathcal{D}_3[d_2, d_1] \parallel s_{(d_3,d_2,d_1)})$;
end

We execute the $\mathcal{OPA}_3$ algorithm where the whole hard disk is scanned only once from s_0 to s_{N-1}, which is independent of the dimensionality. $\{v\_\mathcal{D}_1[d_3, d_2], v\_\mathcal{D}_2[d_3, d_1], v\_\mathcal{D}_3[d_2, d_1] \mid 0 \leq d_3 \leq R_3, 0 \leq d_2 \leq R_2, 0 \leq d_1 \leq R_1\}$ is stored securely somewhere for later comparison.

3.3 Integrity check

If the integrity of a whole hard disk needs to be checked, $\mathcal{OPA}_3$ should be run again to calculate all hash values and compare them with all stored ones. The mismatched hash values will be picked out. If there exist hash values satisfying $v\_\mathcal{D}_1[d_3, d_2] \neq v\_\mathcal{D}_1'[d_3, d_2]$, $v\_\mathcal{D}_2[d_3, d_1] \neq v\_\mathcal{D}_2'[d_3, d_1]$, and $v\_\mathcal{D}_3[d_2, d_1] \neq v\_\mathcal{D}_3'[d_2, d_1]$, sector $s_{(d_3,d_2,d_1)}$ is considered to be the modified or bad one. This algorithm is denoted as $\mathcal{ICHD}_3$ and is shown in Algorithm 4.

If the integrity of a certain sector s_j is under checking, the forensics investigator should first locate the three hash chains that contain the sector and re-calculate the corresponding three hash chain values by scanning the hard disk to read the

Algorithm 4: $\mathcal{ICHD}_3$, integrity check of hard disk for 3D

output: $\{(d_3, d_2, d_1)\}$
Call $\mathcal{OPA}_3$;
if there exist hash values satisfying $v\_\mathcal{D}_1[d_3, d_2] \neq v\_\mathcal{D}_1'[d_3, d_2]$, $v\_\mathcal{D}_2[d_3, d_1] \neq v\_\mathcal{D}_2'[d_3, d_1]$, or $v\_\mathcal{D}_3[d_2, d_1] \neq v\_\mathcal{D}_3'[d_2, d_1]$ **then**
  return all satisfied $\{(d_3, d_2, d_1)\}$;
else
  return $\varnothing$;
end

sectors locating in any of the three chains. Any one of the three matches is a valid proof for the integrity of the sector. Only if all of the three hash values do not match the previous ones calculated in Section 3.2, can the sector not be proved whether it has been modified. This algorithm is denoted as $\mathcal{ICS}_3$ and is shown in Algorithm 5. For example, for sector s_9 mapped into $s_{(0,1,2)}$, it cannot be identified whether s_9 has been modified when all of the three newly calculated hash values are unequal to the previously stored ones, i.e., $v\_\mathcal{D}_1[0,1] \neq v\_\mathcal{D}_1'[0,1]$, $v\_\mathcal{D}_2[0,2] \neq v\_\mathcal{D}_2'[0,2]$, $v\_\mathcal{D}_3[1,2] \neq v\_\mathcal{D}_3'[1,2]$. We will discuss the probability, which is negligible for a sector losing the integrity evidence, in Section 4.

Although the kD scheme can be extended from the concrete 3D case easily, we append the general algorithms $\mathcal{SDA}_k$ and $\mathcal{OPA}_k$ for kD (arbitrary k) in the Appendix.

4 Analysis and observations of the kD hashing scheme

After designing the above three algorithms to distribute sectors in k dimensions, we need to evaluate the scheme according to two criteria: (1) the storage of hash values generated as integrity evidence; (2) the probability for a sector losing integrity evidence.

Since there are $N^{(k-1)/k}$ hash values in each dimension where N is the number of total sectors in a hard disk, the total number of hash values in k dimensions is

$$N_{\text{hash}}(k) = k \times N^{(k-1)/k}. \tag{1}$$

To calculate the probability for a sector, which may be wrongly judged, we must know how often

Algorithm 5: $\mathcal{ICS}_3$, integrity check of a sector for 3D

input : j
output: Yes (integrity is kept) or No
$(d_3, d_2, d_1) = \mathcal{SDA}_3(j)$;
Set $v\_\mathcal{D}_1'[d_3, d_2]$, $v\_\mathcal{D}_2'[d_3, d_1]$, and $v\_\mathcal{D}_3'[d_2, d_1]$ to the initial values;
foreach d_1 $(0 \leq d_1 \leq R_1)$ **do**
  $i = \mathcal{SDA}_3^{-1}(d_3, d_2, d_1)$;
  $v\_\mathcal{D}_1'[d_3, d_2] = f(v\_\mathcal{D}_1[d_3, d_2] \,||\, s_i)$;
end
if $v\_\mathcal{D}_1[d_3, d_2] = v\_\mathcal{D}_1'[d_3, d_2]$ **then**
  return Yes;
else
  foreach d_2 $(0 \leq d_2 \leq R_2)$ **do**
    $i = \mathcal{SDA}_3^{-1}(d_3, d_2, d_1)$;
    $v\_\mathcal{D}_2'[d_3, d_1] = f(v\_\mathcal{D}_2[d_3, d_1] \,||\, s_i)$;
  end
  if $v\_\mathcal{D}_2[d_3, d_1] = v\_\mathcal{D}_2'[d_3, d_1]$ **then**
    return Yes;
  else
    foreach d_3 $(0 \leq d_3 \leq R_3)$ **do**
      $i = \mathcal{SDA}_3^{-1}(d_3, d_2, d_1)$;
      $v\_\mathcal{D}_3'[d_2, d_1] = f(v\_\mathcal{D}_3[d_2, d_1] \,||\, s_i)$;
    end
    if $v\_\mathcal{D}_2[d_3, d_1] = v\_\mathcal{D}_2'[d_3, d_1]$ **then**
      return Yes;
    else
      return No;
    end
  end
end

a sector changes in the real case. Although generally physical defect is likely to affect more than one sector, we assume each sector has an independent probability p of being a bad sector after some time for discussion. Obviously, p is very small. For the probability analysis, we define $P_{\text{f}}(k)$ as the probability for a sector s_{i_0,j_0,k_0} that we fail to judge the integrity, provided that each sector in the whole hard disk becomes bad with probability p using the kD scheme (Jiang *et al.*, 2008):

$$P_{\text{f}}(k) = [1 - (1-p)^{N^{1/k}-1}]^k (1-p). \tag{2}$$

Extension to m blocks: Although increasing the dimensionality k decreases $P_{\text{f}}(k)$, the number of hash values $N_{\text{hash}}(k)$ also increases. It is therefore necessary to examine how $P_{\text{f}}(k)$ may be reduced while $N_{\text{hash}}(k)$ is also reduced. One straightforward way is to divide the N sectors into m blocks ($m \geq 1$),

and apply the kD hashing scheme to each individual block. This strategy is simple and effective. Even in the 1D case, by setting m to N, the probability $P_{\mathrm{f}}(k)$ can be reduced to 0 with $N_{\mathrm{hash}}(k)$ set to $N!$. This is the absolute minimum value of $P_{\mathrm{f}}(k)$. Therefore, it is necessary to consider the combined effect of the dimensionality k and the number of blocks m.

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

$$N_{\mathrm{hash}}(k,m)=mk(N/m)^{(k-1)/k}. \tag{3}$$

Upon substituting m for the number of total disk sectors N, the failure probability for the kD hashing scheme is given by

$$P_{\mathrm{f}}(k,m)=[1-(1-p)^{(N/m)^{1/k}-1}]^k(1-p). \tag{4}$$

4.1 Analysis

Fig. 2 shows the variation of the number of hash values stored ($N_{\mathrm{hash}}(k,m)$) versus the number of blocks divided (m) for dimensions $k=1,2,3,4$ using Eq. (3) with a fixed number of sectors ($N=4.88\times10^8$ for a 250 GB hard disk). Fig. 3 presents the probability for a sector to be wrongly judged as bad ($P_{\mathrm{f}}(k,m)$) with varying m for dimensions $k=1,2,3,4$ using Eq. (4), with $N=4.88\times10^8$ and a fixed independent probability of being a bad sector ($p=10^{-5}$). Increasing k while keeping m fixed yields a lower failure probability $P_{\mathrm{f}}(k,m)$. When k is increased by 1, $P_{\mathrm{f}}(k,m)$ drops exponentially by a value of approximately p. This fact can be partially explained by simplifying Eq. (4). Using the fact that $(1-e)^x$ can be approximated by $1-ex$ when e is very small and m is an integer larger than 1, we can approximate $P_{\mathrm{f}}(k,m)$ to $p[(N/m)^{1/k}-1]^k(1-p)$. With further simplification (ignoring the -1), $P_{\mathrm{f}}(k,m)$ is approximated to $p^k(N/m)(1-p)$. Therefore, when N and m are kept constant, every increment in k will result in a reduction roughly by the factor of p in $P_{\mathrm{f}}(k,m)$.

Also, this leads to the observation that even if m is being changed (but not with a huge amount), it is also beneficial to use a higher dimensionality if we want to lower $P_{\mathrm{f}}(k,m)$. This drop of $P_{\mathrm{f}}(k,m)$ due to higher k is parameterized by p. If p is a small value, it will be more favorable to use a higher dimensionality. Note that in the practical situation, the expected number of bad sectors in the hard disk should be small, so p should be very small. For example, when $N=4.88\times10^8$ and $p=10^{-5}$, the expected number of bad sectors is larger than 4.0×10^3. This is a little overestimated. Fig. 4 presents the result similar to Fig. 3 with $p=10^{-10}$, which is more practical.

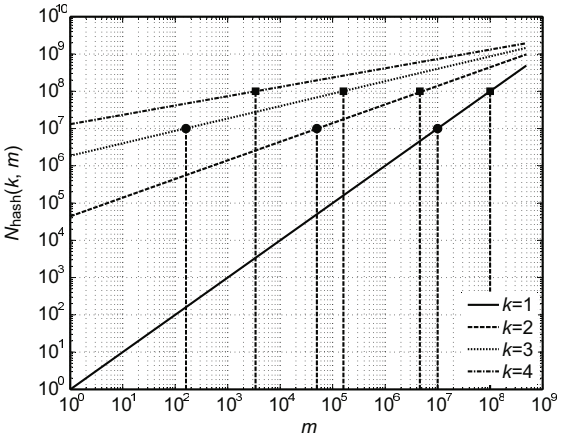

Fig. 2 Number of hash values versus the number of blocks with $N=4.88\times10^8$ for a 250 GB hard disk

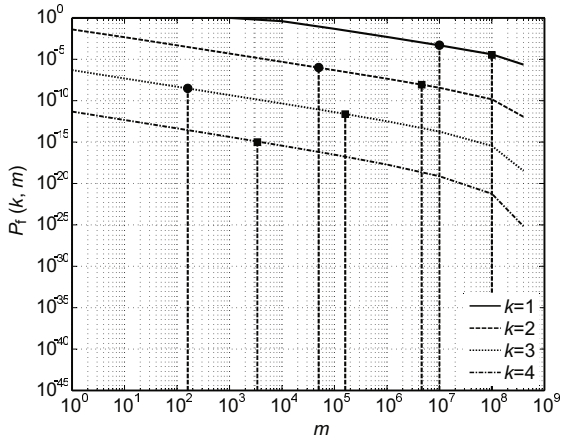

Fig. 3 Failure probability versus the number of blocks with $N=4.88\times10^8$ and a fixed independent probability of being a bad sector $p=10^{-5}$ for a 250 GB hard disk

Then we explore the following problems: Given a fixed $N_{\mathrm{hash}}(k,m)$, which dimensionality should be used to achieve a smaller failure probability $P_{\mathrm{f}}(k,m)$? Using Fig. 2, given a fixed $N_{\mathrm{hash}}(k,m)$, we can find the different numbers of blocks (m) for each dimensionality that will need $N_{\mathrm{hash}}(k,m)$ hash values. From those m values, we can use Fig. 3, or Fig. 4, to find the corresponding $P_{\mathrm{f}}(k,m)$ values for each dimensionality, and decide which dimensionality can offer a lowest $P_{\mathrm{f}}(k,m)$ value.

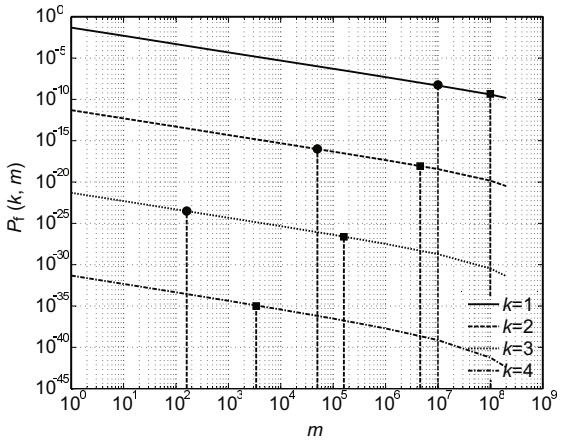

Fig. 4 Failure probability versus the number of blocks with $N = 4.88 \times 10^8$ and a fixed independent probability of being a bad sector $p = 10^{-10}$ for a 250 GB hard disk

Here is an example: Given a fixed $N_{\text{hash}}(k, m) = 10^8$, we find the four squares in Fig. 2 having the same $N_{\text{hash}}(k, m)$, corresponding to $k = 4, 3, 2, 1$. Now record the m values of these four squares, move to Fig. 3, and find four $P_{\text{f}}(k, m)$ values related to these recorded m values corresponding to $k = 4, 3, 2, 1$. Now the corresponding $P_{\text{f}}(k, m)$ values for these four squares will be the different $P_{\text{f}}(k, m)$ values for $k = 4, 3, 2, 1$, with 10^8 hash values. Thus, these four values can be compared.

Upon investigating several different $N_{\text{hash}}(k, m)$ values (e.g., $N_{\text{hash}}(k, m) = 10^7$ or 10^8), we find that it is always better to use a higher dimensionality, provided that $N_{\text{hash}}(k, m)$ is at least the minimum number of hash values needed by that dimensionality (note that a scheme with a higher dimensionality will lead to more hash values stored). Two concrete examples (squares and balls) in Figs. 2–4 illustrate the effect of increasing the dimensionality. In both cases, it is obvious that for the given $N_{\text{hash}}(k, m)$ value, a higher k yields a lower $P_{\text{f}}(k, m)$. Upon comparing the two groups of points (squares and balls), it is apparent that a higher m value can result in a lower $P_{\text{f}}(k, m)$ for the same k.

4.2 Observations

The analysis indicates that the kD hashing scheme is very efficient in reducing the probability $P_{\text{f}}(k, m)$. For example, the failure probability for $m = 10$ blocks (with $p = 10^{-10}$ and $N = 4.88 \times 10^8$) reduces from 4.87×10^{-3} in the 1D scheme of using one hash value for the entire hard disk to 4.65×10^{-33} when 4D hashing is used. This is a dramatic decrease in failure probability. Similar reductions occur for other parameter settings.

Our findings can be summarized in the following recommendations. If the minimization of the failure probability $P_{\text{f}}(k, m)$ is the principal goal and $N_{\text{hash}}(k, m)$ values can be stored, where $N_{\text{hash}}(k) < N$, then it is best to use the highest possible kD hashing scheme. If $N_{\text{hash}}(k, m)$ is close to or larger than N, then the 1D hashing scheme with $P_{\text{f}}(k, m) = 0$ 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 (Garber, 2001; Chow *et al.*, 2005)). The Merkle hash tree (Merkle, 1989) is a low-overhead approach for signing multiple hash values (Wang *et al.*, 2007). Nevertheless, it is important to investigate the effect of the overhead involved in digital signing on the choice of dimensionality.

5 Implementation and evaluation

Two test hard disks, including HD1 with capacity 250 GB (total number of sectors N = 488 392 065) and speed 5400 r/min and HD2 with capacity 60 GB (total number of sectors N = 117 210 240) and speed 4200 r/min, are used in the kD scheme for $k = 3$ and $m = 1$. That is, we implement a 3D scheme without blocking. It is compared with two trivial solutions mentioned in the introduction, 1D and ND schemes where N represents the total number of sectors in the two hard disks. MD5 with 128 bits is chosen as the hash function. The workstation running experimental tests is configured with an Intel® Core™ 2 CPU (E6750 at 2.66 GHz) and 1.97 GB of RAM. We mainly evaluate:

1. The storage of the hash values ($S_{\text{hash}} = 16N_{\text{hash}}(3)$);

2. The memory overhead during running $\mathcal{OPA}_3$ to generate hash values as integrity evidence (M);

3. The time to run $\mathcal{OPA}_3$ to generate hash values as integrity evidence (T_{OPA});

4. The time to run $\mathcal{ICHD}_3$ (T_{ICHD});

5. The time to run $\mathcal{ICS}_3$ (T_{ICS}).

As shown in Table 1, the storage of 28.40 MB hash values using the 3D scheme is acceptable compared with 16 bytes using the 1D scheme, and

Table 1 Experimental results for HD1 (250 GB)

Scheme	S_{hash}	T_{OPA} (s)	T_{ICHD} (s)	T_{ICS} (s)
1D	16 bytes	11 493	11 510	11 496
3D	28.40 MB	16 245	16 255	18.14
ND	7.28 GB	13 526	16 036	0.02

* $N = 488\ 392\ 065$

Table 2 Experimental results for HD2 (60 GB)

Scheme	S_{hash}	T_{OPA} (s)	T_{ICHD} (s)	T_{ICS} (s)
1D	16 bytes	4119	4159	4139
3D	10.98 MB	5248	5261	4.73
ND	1.7 GB	4661	5141	0.02

* $N = 117\ 210\ 240$

is much smaller than the 7.28 GB using the ND scheme. The memory overhead (M) during running $\mathcal{OPA}_3$ is 28.40 MB, the same as the storage. It is a small overhead for normal PCs with a standard configuration. The computational time for calculating hash values in 3D is 16 245 s, which brings a little more overhead than those in 1D and ND, the existing solutions mentioned in Section 1. If a certain sector's integrity must be checked, our 3D scheme takes obvious advantage of 18.14 s compared with 1D because the whole hard disk is scanned to calculate the hash value in the 1D scheme. The comparison among these three parameters of computational time shows that our scheme requires little additional computation overhead. Table 2 shows the similar result.

In summary, compared with the 1D scheme, our 3D scheme is much practical when a sector's integrity must be checked and all other overheads are acceptable. Compared with the ND scheme which usually brings great overhead in storage, our 3D scheme concerns more from the forensics investigators' viewpoint.

6 Conclusions

The kD hashing scheme is a robust technique for verifying the integrity of data stored on hard disk even if some of the sectors suddenly become bad sectors or modified purposely during the investigation or storage. The scheme computes the hash values for each sector in k dimensions; thus, when one or more sectors go bad, it can still verify the integrity of the data in the unaffected sectors with high probability. Experiments show that the scheme is efficient and practical in real situations.

Computer forensics case handling software must handle the problem of hard disk sector integrity properly. This is becoming more and more important as storage size is increasing rapidly. If the computer forensics software had incorporated the kD scheme (with k smaller than the number of disk sectors), the resource requirement of this software will be greatly reduced, and the probability of losing integrity evidence of a disk sector will be greatly reduced. We believe that research results in disk sector integrity checking will greatly benefit the digital investigation.

The deployment of the kD scheme can also discover the illegal modification of integrity information by a wicked computer forensics scientist. This is achieved by providing more than one hash value for each individual sector. Further research efforts are needed to analyze this situation in more detail. Another obvious direction is to design a better scheme to deal with the hard disk integrity checking problem. From a practical point of view, we should consider how to improve this hashing scheme to the situation in which normal users may update their hard disks every day, such that many hash values are needed to be recomputed in order to track the integrity of the hard disks due to the changes in sectors. Since the kD hashing scheme is designed to provide effective integrity proof of a huge amount of data which are prone to be changed partly, it may be extended to other areas to handle similar problems, such as DNA library screening and error correction in communication networks.

References

Chen, B.M., Lee, T.H., Peng, K., Venkataramanan, V., 2006. Hard Disk Drive Servo Systems. Springer, London, p.3-11.

Chow, K.P., Chong, C.F., Lai, K.Y., Hui, L.C.K., Pun, K.H., Tsang, W.W., Chan, H.W., 2005. Digital Evidence Search Kit. 1st Int. Workshop on Systematic Approaches to Digital Forensic Engineering, p.187-194. [doi:10.1109/SADFE.2005.10]

Comito, C., Patarin, S., Talia, D., 2007. PARIS: a Peer-to-Peer Architecture for Large-Scale Semantic Data Integration. Proc. Databases, Information Systems, and Peer-to-Peer Computing, p.163-170. [doi:10.1007/978-3-540-71661-7_15]

Garber, L., 2001. Computer forensics: high-tech law enforcement. *IEEE Comput. Mag.*, **34**(1):22-27. [doi:10.1109/MC.2001.10008]

Gauravaram, P., McCullagh, A., Dawson, E., 2006. Collision Attacks on MD5 and SHA-1: Is This the 'Sword of Damocles' for Electronic Commerce? Auscert Asia Pacific Information Technology Security Conf.: Refereed R&D Stream, p.73-88.

Harbour, N., 2002. dcfldd. Defense Computer Forensics Lab. Available from http://dcfldd.sourceforge.net

Hussain, O.K., Dillon, T.S., Chang, E., Hussain, F., 2010. Transactional risk-based decision making system in e-business interactions. *Int. J. Comput. Syst. Sci. Eng.*, **25**(1):15-25.

Jiang, Z.L., Hui, L.C.K., Chow, K.P., Yiu, S.M., Lai, P.K.Y., 2007. Improving Disk Sector Integrity Using 3-Dimension Hashing Scheme. Int. Workshop on Forensics for Future Generation Communication, p.141-145.

Jiang, Z.L., Hui, L.C.K., Yiu, S.M., 2008. Improving Disk Sector Integrity Using *k*-Dimension Hashing. Advances in Digital Forensics IV, p.87-98. [doi:10.1007/978-0-387-84927-0_8]

Kornblum, J., 2006. Identifying almost identical files using context triggered piecewise hashing. *Dig. Invest.*, **3**(Supplement 1):91-97. [doi:10.1016/j.diin.2006.06.015]

Law, F.Y.W., Lai, P.K.Y., Jiang, Z.L., Ieong, R.S.C., Kwan, M.Y.K., Chow, K.P., Hui, L.C.K., Yiu, S.M., Chong, C.F., 2008. Protecting Digital Legal Professional Privilege (LPP) Data. 3rd Int. Workshop on Systematic Approaches to Digital Forensic Engineering, p.91-101. [doi:10.1109/SADFE.2008.19]

Mead, S., 2006. Unique file identification in the National Software Reference Library. *Dig. Invest.*, **3**(3):138-150. [doi:10.1016/j.diin.2006.08.010]

Merkle, R.C., 1989. A Certified Digital Signature. Advances in Cryptology, p.218-238.

NIST (National Institute of Standards and Technology), 2004. National Software Reference Library (NSRL). Available from http://www.nsrl.nist.gov

Schroeder, B., Gibson, G.A., 2007. Disk Failures in the Real World: What Does an MTTF of 1 000 000 Hours Mean to You? 5th USENIX Conf. on File and Storage Technologies, p.1.

Wang, M., Li, L., Yiu, S.M., Hui, L.C.K., Chong, C.F., Chow, K.P., Tsang, W.W., Chan, H.W., Pun, K.H., 2007. A Hybrid Approach for Authenticating MPEG-2 Streaming Data. Int. Conf. on Multimedia Content Analysis and Mining, p.203-212. [doi:10.1007/978-3-540-73417-8_27]

Appendix: a general *k*D scheme

Since a kD scheme can be extended by the specific 3D case, we just list the sector distribution algorithm $\mathcal{SDA}_k$ and the one-pass algorithm $\mathcal{OPA}_k$ in Algorithms A1 and A2 without detailed description.

Similar to Theorem 1, we have the following theorem:

Theorem A1 Function $\mathcal{SDA}_k$ is a function from a set $X=\{j \mid 0 \le j \le N-1\}$ to a set $Y=\{(d_k,\cdots,d_1) \mid 0 \le d_k \le R_k,\cdots,0 \le d_1 \le R_1\}$, where $R_1,\cdots,R_k$ are decided by N, with the property that, for every $(d_k,\cdots,d_1)$ in Y, there is exactly one j in X such that $\mathcal{SDA}_k(j)=(d_k,\cdots,d_1)$. Or $\mathcal{SDA}_k$ is a bijection.

Algorithm A1: $\mathcal{SDA}_k$, sector distribution algorithm for kD

input: j. **output**: $(d_k,d_{k-1},\cdots,d_1)$.
if $j=0$ **then**
  $(d_k,d_{k-1},\cdots,d_1)=(0,0,\cdots,0)$; Exit;
end
$L=\left\lfloor j^{1/k}\right\rfloor$;
if $L^k < j+1 \le L^k+L^{k-1}$ **then**
  $i=j-L^k$; $d_1=L$;
  foreach m $(2 \le m \le k)$ **do**
    $d_m=i/L^{m-2}\%L$;
  end
else if
$L^k+L^{k-1} < j+1 \le L^k+L^{k-1}+L^{k-2}(L+1)$
then
  $i=j-(L^{k-1}+L^{k-2})$;
  $d_1=i\%(L+1)$; $d_2=L$;
  foreach m $(3 \le m \le k)$ **do**
    $d_m=i/(L^{m-3}(L+1))\%L$;
  end
...
else if $L^k+\sum_{x=1}^{t-1}L^{k-x}(L+1)^{x-1} < j+1 \le \sum_{x=1}^{t}L^{k-x}(L+1)^{x-1}$ **then**
  $i=j-L^k-\sum_{x=1}^{t-1}L^{k-x}\ (L+1)^{x-1}$;
  foreach m $(1 \le m \le t-1)$ **do**
    $d_m=i/(L+1)^{m-1}\%(L+1)$;
  end
  $d_t=L$;
  foreach m $(t+1 \le m \le k)$ **do**
    $d_m=i/(L^{m-t}(L+1)^{t-2})\%L$;
  end
...
else if
$L^k+\sum_{x=1}^{k-1}L^{k-x}(L+1)^{x-1} < j+1 \le (L+1)^k$
then
  $i=j-L^k-\sum_{x=1}^{k-1}L^{k-x}(L+1)^{x-1}$;
  foreach m $(1 \le m \le k-1)$ **do**
    $d_m=i/(L+1)^{m-1}\%(L+1)$;
  end
  $d_k=L$;
end

Algorithm A2: $\mathcal{OPA}_k$, one-pass algorithm for kD

input: $\{s_j \mid j=0,1,\cdots,N-1\}$
output: $\{v\_\mathcal{D}_t[\,] \mid t=1,2,\cdots,k\}$
Set all $\{v\_\mathcal{D}_t[\,] \mid t=1,2,\cdots,k\}$ to the initial values;
foreach sector s_j where $j=0,1,\cdots,N-1$ **do**
  $(d_k,d_{k-1},\cdots,d_1)=\mathcal{SDA}_k(j)$;
  $v\_\mathcal{D}_t[d_k,\cdots,d_{t+1},d_{t-1},\cdots,d_1]$;
end