Hard Disk Integrity Check by Hashing with Combinatorial Group Testing

Junbin Fang, Zoe L. Jiang, S.M. Yiu, Lucas C.K. Hui
Department of Computer Science
The University of Hong Kong
Email: jbfang,ljiang,smyiu,hui@cs.hku.hk

***Abstract*—In this paper, we describe the problem of verifying the integrity of a hard disk especially for forensics investigation after the computer of a suspect has been seized. Existing solutions do not provide a satisfactory solution to solve the problem. They either require a huge amount of storage to store the hash values of the sectors or may not be able to cope with the situation in an effective way in case some sectors have been modified (e.g. become bad sectors or deleted due to being part of the Legal Professional Privilege items). We introduce to use Thierry-Mieg[15]'s combinatorial group testing scheme, which seems to be an unrelated topic, to design a scheme to compute the hash values for the sectors of a hard disk. The storage for hash values in our scheme can be significantly fewer than the best existing solution while requiring similar amount of execution time. And our scheme can accurately point out the sectors which have been modified while existing solutions cannot guarantee this.**

I. Introduction

With the rapid development of electronic commerce, digital forensics have 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 modify, one of the biggest problems is how to effectively maintain the integrity of digital evidence. In this paper, the following problem will be considered. Given a hard disk seized from a suspect's computer, how to effectively check the integrity of the sectors inside the hard disk to guarantee that the content of the hard disk remains the same as before and has not been modified or tampered so that the evidence found inside the hard disk can be accepted by courts. Otherwise, the suspect can easily challenge the validity of the collected evidence. Sometimes, the problem can be more complicated that if the modified or tampered sectors are not related to the evidence, we may try to see if the integrity of this evidence itself can still be verified.

To maintain the integrity of a piece of data, the standard technique is to first compute a hash value of the data and store it somewhere securely. Then, recompute the hash value again when the investigation is needed. If both values agree, the data are believed to be unchanged. For example, in some of the digital forensics tools (e.g., [1], [2]), they use the chained hashing scheme to verify the integrity of data on hard disks. In details, they compute one single chained one-way hash value of all the disk sectors in a specific order. Then store the signed hash value in a secure location for later comparison. However, the technique can only provide a YES/NO answer about the integrity of the whole hard disk which is not good enough. In some cases when some of the sectors may suddenly turn into bad after a long time (say a few months), it may be useful to know which sectors affect the integrity and whether the modified sectors are related to the evidence or not. In other cases when the suspect is allowed to modify or delete some of the sectors on his own (e.g. files classified as Legal Professional Privilege data [3]), we may try to see if the integrity of the evidence is not affected. So, the approach of storing one single chained hash value is less attractive.

The other extreme technique is to compute a hash value for each sector, then sign and store all these signed hash values for later comparison. However, there will be too many hash values to be stored. For example, for a hard disk with capacity of 160GB, there will be 3.5×10^8 hash values to be stored. It is about 5GB storage if each hash value has 128 bits. With the size of hard disk getting larger and larger, this approach is not appropriate either.

A. *Related work*

There are other approaches proposed by the researchers to solve this problem. Kornblum [7] described Context Triggered Piecewise Hash (CTPH) scheme to identify modified versions of known files even if data have been inserted, modified, or deleted in the new files. Although it may be possible to make it work for the sectors in a hard disk (for a sector becoming bad can be considered as a part of a file being modified), the CTPH scheme requires a long running time of $O(n \log n)$ where n is the data size, which implies a significant high overhead. For example, when it is applied to a hard disk with huge size such as 200G bytes, the execution time is about 40 times more than reading the whole hard disk once.

Recently, Jiang et al. proposed a CHS scheme [4] and an improved k-$\mathcal{D}$ scheme [5] to check the integrity of the sectors in a hard disk by calculating more than one hash value for each sector so as to increase the chance of it being verified successfully even if there are bad/modified sectors. In terms of execution time and storage for hash values, the k-$\mathcal{D}$ scheme is more efficient than the above two techniques. However it fails to verify some sectors' integrity with a certain probability, especially when more and more sectors are modified. It may be unacceptable in practice. We shall discuss it in detail in section II.

Goodrich et al. [6] designed a novel scheme to organize the indexing structures of several fundamental data structures using Combinatorial Group Testing (CGT) algorithm, such that alterations from an original version can be detected and identified. However, to meet their requirement of encoding both the CGT matrix as well as the stored hash values within the data structure without extra storage, their CGT construction algorithm may not always produce a correct matrix (although with low probability). This property is not that appropriate in the computer forensics context. Besides, the number of tests required is $6(d+1)(d+2)\lceil \ln n \rceil$ where d is the maximum number of errors and n is the number of total items. If d is large, the number of tests is huge. So is the storage.

B. CGT background

Combinatorial Group Testing (CGT) was first proposed by Dorfman[8] during World War II when blood samples of millions of draftees were subject to identical analyses in order to detect a few thousand cases of syphilis. In the original scheme, tester first extracts a few drops of blood samples in a group and pools them together, then tests the mixed sample to discover the existence of syphilis antigen in this group. When the outcome of the group test is negative, all samples in the group are good (disease-free). Otherwise, it implies that some samples inside this group are defective and further testing on them are necessary. Using this approach, the cost of identifying which of n blood samples are tainted can be significantly decreased by applying tests to judiciously chosen subsets of the samples - given any constant upper bound on the number of tainted samples, a logarithmic (in n) number of tests suffices.

Due to the high efficiency of CGT, in the 1960s, Sobel and Groll [9] revived the interest in it by giving many industrial applications in detecting chemical leakage and electrical blocking. At present, the applications of CGT are extended to a variety of fields including DNA library screening, communication, coding theory (e.g. [10], [11]).

The key issue of CGT algorithm is how to construct the testing groups, i.e. how to choose subsets of the set into groups, to minimize the total number of tests with guaranteed ability of locating the problematic cases (defectives). There are several known methods for the construction of CGT algorithm, e.g. set packing designs [12], direct constructions[13]. In [14], Ngo and Du have made a survey on CGT algorithms and have given a taxonomy of CGT constructions. In 2006, an efficient non-adaptive combinatorial pooling construction, the Shifted Transversal Design (STD) was proposed by Thierry-Mieg[15], which is highly flexible and can be tailored to function robustly. It will be used in our application.

C. Our Contributions

In this paper, we adapt CGT algorithm with the STD construction to improve the hard disk integrity problem in the efficiency of verification by saving the cost of storage or calculation. Besides, bad (or modified) sectors can be accurately identified if needed. It is necessary to claim that the paper does not focus on developing novel CGT algorithm theoretically, but singling out the most appropriate CGT algorithm and adapting it to in the computer forensics area to help solve the integrity checking problem.

The following shows the details of our contributions.

- We adapt Thierry-Mieg[15]'s CGT algorithm for the verification of the integrity of the sectors in a hard disk integrity. Our design is based on the STD construction which is the most efficient approach for designing test groups in a non-adaptive CGT algorithm.
- The scheme can significantly reduce the cost of verifying the integrity of a hard disk by reducing the storage needed for hash values (that is, the number of hash values to be computed) by more than 10 times as the 3-$\mathcal{D}$ scheme while maintaining a similar executing time.
- The scheme provides a guaranteed ability of identifying all sectors accurately and detecting the sectors which have been modified provided the number of such sectors is no more than the threshold we set in the design.
- We compare the performance of our CGT scheme with 3-$\mathcal{D}$ scheme. The results show that our scheme is more efficient than the 3-$\mathcal{D}$ scheme. We also show that this scheme can be used by normal users if they want to keep track the integrity of their hard disk frequently. The basic idea is to recompute some hash values each day after they modify some of the sectors.

D. Outline

The organization of the rest of the paper is as follows. k-$\mathcal{D}$ [5] scheme and its limitations will be reviewed in Section II. CGT construction by STD, especially the parameter design, is described in Section III. Our experimental results are shown and explained in Section IV, with a comparison to the results of the 3-$\mathcal{D}$ scheme [5]. Section V concludes the paper.

II. Review of k-$\mathcal{D}$ scheme

The k-$\mathcal{D}$ scheme can be regarded as a trade-off between the approaches of storing only one hash value for all sectors and storing a hash value of each sector to address the issue for checking the integrity of a hard drive even if some of the sectors become bad sectors without storing the hash value for each sector. For a hard disk with a total of N sectors which are ordered in a k-dimensional plane, instead of computing one hash for all sectors, the scheme computes a hash value for each sector chain in different dimensions to increase the chance of a sector being verified successfully even if there are modified sectors on some of its chains. Refer to [5] for more details. We give an analysis of its performance and highlight the limitations of this scheme here.

First, although the number of hash values to be stored by this scheme is already substantially fewer than the approach of storing one hash value per sector, it still requires quite a large number of hash values for small value of k (e.g. $k = 3$

for practical use). $S_k = k \times N^{(k-1)/k}$ shows the number of hash values required by the scheme. $C_k = k \times N$ calculate the number of hash functions to be executed. When one sector is updated, the scheme also requires to recompute the number of hash values by $Cr_k = k \times N^{1/k}$.

For example, given a hard disk of 1×10^9 sectors, which is about 479GB if each sector contains 512 bytes, the 3-$\mathcal{D}$ scheme needs a storage of 3×10^6 hash values and 3×10^9 hash calculations.

The other limitation of the scheme is that it may fail to recognize an unmodified disk sector correctly due to other bad sectors. For example, 3-$\mathcal{D}$ scheme is used where each sector will be assigned to three different chains. If an unmodified sector is assigned to three chains in which there is at least one of the sectors in each chain that has been modified, the hash values of these three chains will not be equal to the stored hash values. Thus, there is no way to verify the integrity of this sector. The probability to have such an unmodified sector (but not verifiable) is $Pf_k = [1 - (1 - N_b/N)^{(N^{1/k}-1)}]^k$, where N_b is the number of bad sectors.

In practical applications, it is desirable to have a better scheme even if Pf_k is small since the sectors affected may relate to an important digital evidence. To conclude, the k-$\mathcal{D}$ scheme has its limitations and improvement in efficiency and reliability is desirable.

III. Hard disk integrity checking using CGT approach

In this section, we first introduce the idea of adapting CGT into the hard disk integrity checking problem. Then by reviewing the selected construction - Shifted Transversal Design (STD) [15], some specific concerns in terms of integrity checking are also considered.

A. Adapting CGT to solve the hard disk integrity problem

Before looking into the hard disk integrity problem, let us observe the following simple example.

Assume that a set C under test contains five items: $\{1,2,3,4,5\}$ and no more than one item in the set is defective. We can design a pattern of group testing, which can be represented by a 3×5 binary matrix as follows.

$$M = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix},$$

where each column corresponds to an item of C and each row corresponds to a group test and each cell (i, j) is 1 if and only if item j is included in the test case i. In other words, in our example, for test case 1, we include items 4 and 5 together. Therefore, with this testing pattern, if the group tests on $\{1,3,5\}$ (test case 3) and $\{2,3\}$ (test case 2) show that one of the items is defective while the test on $\{4,5\}$ (test case 1) indicates all items are normal, we can conclude that item 3 is the defective one. One can check for other possibilities that these three test cases suffice to deduce the defective item. In this case, only 3 group tests but not 5 individual tests are needed to find out the defective (problematic) item.

Briefly speaking, CGT is mainly applied to situations where it is very expensive to test every single item individually of a large number of items and it is necessary to pinpoint the defective items precisely. When CGT is introduced into our scheme to deal with the hard disk integrity problem, an analogy should be made first. Assuming that an unmodified sector in the hard disk is a normal item and a modified or corrupted (bad) sector is a defective item, the cryptographic hash of a subset of n sectors of the hard disk is analogous to a mix of blood drops from a subset of n blood samples, and comparing such a hash to what it is supposed to be, for example, a stored hash, is analogous to applying a blood test to the mix of blood drops. A mismatch between the computed hash value and the stored hash value indicates that there is at least one corrupted (bad) sectors in the subset for that hash. From the results of all the comparisons, CGT algorithm can precisely identify each sector as modified or not.

Applying CGT in the hard disk integrity problem will have some advantages (or improvements). On one hand, since the number of stored hash values equals the number of group tests, the storage required for storing the hash values can be greatly decreased by applying CGT. On the other hand, CGT can provide a more accurate indication on which sectors are modified while the 3-$\mathcal{D}$ scheme may loose the integrity checking evidence of a certain sector if all the three hash chains of each dimension for the sector are tainted by other bad sectors on these chains.

B. CGT Construction by Shifted Transversal Design (STD)

CGT can be described more mathematically as an especially efficient algorithm aiming at the problem to identify relatively rare events from a large collection of items by applying basic yes-or-no tests. Consider a set of n items which can only be true or false, represented by n Boolean variables. If we call a “pool”(test) a subset of variables and assume that at most d variables are true, then the goal of CGT is to build a set of t pools, where t is small compared to n, such that by testing the values of the t pools, one can unambiguously determine the values of the n variables. Practically, a CGT algorithm with n items and t pools can be represented by a $t \times n$ binary matrix where each cell (i, j) has a 1-entry if and only if item j is contained in test i. Therefore, the pooling design of CGT can be regarded as the problem constructing such a testing matrix. In our scheme, STD, a novel construction approach for CGT proposed in 2006, is used to provide higher efficiency than other existing construction methods.

For ease of discussion, we recall the description of the technical material on how to design the test groups (that is, how to group the sectors into chains) Thierry-Mieg[15].

1) Construction algorithm of STD: A STD-based pool construction starts with the specification of the number of total items (n) and the maximum number of expected defective items (d). These input parameters (n, d) are used to choose the design parameters of the STD construction algorithm q and k, where q must be a prime number. More precisely, STD is a layered construction with k layers, each of size $q \times n$. After the design parameters are selected, the construction algorithm will produce a $t \times n$, $0-1$ matrix $M = STD(n;q;k)$, where $t(= q \times k)$ rows represent the group tests to be performed and n columns represent the items (sectors) to be tested. Since each item appears only once in each layer, each of the n columns has k 1's in it and each row has usually $\lceil n/q \rceil$ 1's in it. While a detailed description and proof of the construction is available in the original STD paper [15], we review how the algorithm works here.

Given the design parameters q and k, the matrix $M = STD(n;q;k)$ can be constructed as follows. The STD has a layered construction consisting of k layers of $q \times n$ boolean matrices. For all $j \in \{0, \cdots, k-1\}$, let M_j be a $q \times n$ boolean matrix representing layer $L(j)$, with columns $C_{j,0}, \cdots, C_{j,n-1}$.

Let the circular shift operator, σ_q, be defined as for all $(x_1, \cdots, x_q) \in \{0,1\}^q$, $\sigma_q \left[x_1 \ x_2 \ \cdots \ x_q\right]^T = \left[x_q \ x_1 \cdots \ x_{q-1}\right]^T$ and $C_{0,0} = \left[1 \ 0 \ \cdots \ 0\right]^T$. Note that σ_q is a cyclic function and when applied q times maps $\{0,1\}^q$ onto itself, $\sigma_q^s \left[x_1 \ x_2 \ \cdots \ x_q\right]^T = \left[x_1 \ x_2 \ \cdots \ x_q\right]^T$, $s = q$. To design a layer $L(j)$, for all $i \in \{0, \cdots, n-1\}$, construct $C_{j,i} = \sigma_q^{s(i,j)} C_{0,0}$ where if $j < q : s(i,j) = \sum_{c=0}^{\Gamma} j^c \lfloor i/q^c \rfloor$, if $j = q : s(i,j) = \lfloor i/q^\Gamma \rfloor$. The layers $L(j)$ are put together to form M by $STD(n;q;k) = \bigcup_{j=0} L(j)$.

2) Choosing design parameters of STD: The detailed procedures of mapping the experimental parameters n and d to the design parameters q and k are listed below.
(1) Choose a prime number q, with $q < n$. Start with the smallest prime, 2.
(2) Find the *compression power*, $\Gamma(q,n) = min\{\gamma | q_{\gamma+1} \geq n\}$, therefore $\Gamma = \lceil \log n / \log q \rceil - 1$. Set $k = d\Gamma + 1$.
(3) Check if this choice of q and k satisfies the *guarantee* requirements of identifying d defective elements, using the inequality, $k \leq q + 1$.
(4) If the inequality is satisfied, continue to step (5), else choose the next prime in step (1) and repeat (2) and (3).
(5) For each q, find its corresponding compression power Γ that satisfies $q \geq n^{1/(\Gamma+1)}$, and calculate the number of tests (t) needed by each (q,k) pair from $t = q \times k$.
(6) Choose q and k to produce the desirable number of tests.

In general testing problem, the most desirable number of tests is the least one in numbers produced by all possible combinations of q and k. Nevertheless, when the application is on the hard disk integrity problem, a special consideration on the numbers should be taken since the hard disk integrity problem has a different setting: we need to consider minimizing the number of hash values as well as minimizing the time to calculate all hash values. In the traditional application of blood testing case, a test of a group of ten individuals has the same (computational) costs as a group of three individuals. But in hash value calculation, the computational cost of calculating a hash value of ten sectors will be more than that of three sectors. In Section IV, three sets of design parameters of STD are demonstrated with experimental results for comparison.

3) Decoding algorithm of STD: This subsection shows how to interpret the testing results. That is, for each group (chain of sectors), we compare the hash value with the stored value, then depending on whether these two values agree, we can identify which sector remains unchanged and which sector has been modified (or turned as a bad sector).

With the designed pooling pattern, M, all the items (sectors) under test can be tagged by STD according to the following decoding algorithm: all the sectors present in at least one normal pool (the test result shows that the hash values matched for the group of sectors) are tagged normal (that is, the sectors are not modified); any variable (sector) present in at least one defective pool (in which the hash values of the group do not match) where all other variables (sectors) have been tagged normal, is tagged defective (modified). The STD algorithm guarantees that the pooled design will be able to correctly identify up to d defective (modified) items. STD is able to provide such guarantees because it uses a combinatorial procedure to ensure that no two items (sectors) are pooled (grouped) together more than a minimum number of times, to prevent ambiguous decoding results. Also, the number of items pooled in each test is roughly the same, ensuring the computational required for each chain is similar.

It is worthy to remark that if there exist more than d defective modified sectors, some sectors may not be tagged while all tags produced by the above algorithm are still correct. These untagged items may include both good items and defective items. Such a situation can be detected by STD and once it is detected some further operations need to be taken to check whether any sector of evidence is included in the untagged items or to retest these items if completeness is sought. Fortunately, with STD's well-chosen pooling design, untagged items should only occur when the number of defective items is much larger than expected maximum d. Furthermore, it has been shown [16] recently that STD is also capable of detecting much larger number of defective items than that it guarantees, with a more sophisticated decoding algorithm. In other words, using CGT, we will not treat an unmodified sector as modified or vice versa as in the 3-$\mathcal{D}$ scheme.

IV. Experimental results

We perform a series of experiments to test the performance of CGT and compare it with that of 3-$\mathcal{D}$ scheme in terms of (1) the number of hash values to be stored and (2) the time required to compute the hash values. Also, we show that the CGT scheme is feasible also for normal users to keep track the integrity of their hard disks by keeping the hash values of the sectors and recompute those hash values every day for those chains in which there are modified sectors. We developed our CGT scheme using STD construction, as well as the 3-$\mathcal{D}$ scheme for comparison. We measure the costs of the actual storage required for storing hash values for integrity check, the actual running time for computations to generate the values and the additional time for re-computations in the case that there is a requirement to re-compute hash values when only some sectors are updated legally.

Two test hard disks are used including HD #1 with capacity of 55GB (60,011,642,880 bytes, 117,210,240 sectors) and speed of 4,200 rpm, and HD #2 with capacity of 232GB (250,056,737,280 bytes, 488,392,065 sectors) and speed of 5,400 rpm. The hardware of our experimental environment includes a PC configured with an Intel® Core™ 2 CPU (E6750 at 2.66GHz) and 1.97 GB RAM.

A. Experimental parameters and testing matrices for CGT

Following the procedures described in STD algorithm, set the parameters n and d as 10^6 and 1, respectively. It means that all sectors of a hard disk will be divided into smaller blocks, each of which contains 10^6 sectors, and the number of bad sectors contained in each block is expectedly at most 1, i.e., the maximum probability for a block which contains bad sector is 10^{-6}. Note that this probability is far more overestimated than the common one, 10^{-9}, which is in the range used previously in the 3-$\mathcal{D}$ scheme [5].

As described in section III.B, with same experimental parameters input, different sets of design parameters can be chosen for STD to accommodate to various applications. Since the performance of CGT algorithm depends on the matrices (how the sectors are grouped into chains), it is essential to study the effect of different parameters and the corresponding matrices in our application. Thus, we choose three sets of STD parameters to construct matrices for some practical considerations. Using the chosen parameters, the scheme can handily construct a CGT matrix using STD construction algorithm. Note that the testing matrix needs to be constructed only once which will be kept unchanged for the further use of distributing sectors into test groups. The selected design parameters and the corresponding testing matrices are:
(M_a) $n = 10^6; q_a = 7; k_a = 8; M_a = STD(10^6; 7; 8)$;
(M_b) $n = 10^6; q_b = 101; k_b = 3; M_b = STD(10^6; 101; 3)$;
(M_c) $n = 10^6; q_c = 1511; k_c = 2; M_c = STD(10^6; 1511; 2)$.

Once the testing matrix of CGT is prepared, it can be applied to each block of the hard disk to calculate hash values. The cost of storage and calculations can also be deduced from the matrix. Given the number of total sectors in the hard disk N, the number of hash values to be stored per 10^6 sectors will be $q \times k$ and the total number of hash values to be stored for the whole hard disk will be $q \times k \times \lceil N/n \rceil$ for CGT with matrix $M = STD(n; q; k)$. In practice, q and k are relatively small constants. So the number of hash values is reasonable and much smaller than [6] when d is large. In the 3-$\mathcal{D}$ scheme, these two numbers will be $3 \times N^{(3-1)/3}$ and $3 \times N$ for the whole hard disk. An additional question is that if there are several sectors modified, how many hash values will be re-computed and how many sectors will be read again and taken into re-computation. For CGT, updating one sector will lead to a re-computation of $k \times (n/q)$ hash calculations, while for the 3-$\mathcal{D}$ scheme, the number of required hash calculations is $3 \times N^{1/3}$ for the whole hard disk.

Before we present the experimental results, we try to analyze the three different settings. In the following, we only consider the amount of hash values to be stored and the number of hash computations to be calculated per 10^6 sectors. For M_a, the constructed matrix has only 51 rows and requires the minimum storage of only 51 hash values for every 10^6 sectors. The shortcoming of M_a is that it has 8 layers and needs 8×10^6 hash calculations for every 10^6 sectors. So, the setting of M_a is suitable for the situations where storage for hash values is the factor we concern the most. On the contrary, M_c has the fewest number of layers and requires the minimum computation (only 2×10^6 hash calculations for every 10^6 sectors), while it requires the maximum storage of hash values ($3,022$ hash values per 10^6 sectors). M_c is considered to be applicable in situations where sectors are often changed, such as the sectors will be changed in a daily basis when the scheme is extended to normal users. As a tradeoff of storage versus calculation, the requirement for storage and computation of M_b is in the middle of that for the other two cases with 303 hash values to be stored and 3×10^6 hash values to be computed per 10^6 sectors.

B. Results and discussion

The two hard disks are tested with M_a, M_b and M_c separately and experimental results include absolute time (T_a), computation time (T_c) and actual storage (S) as shown in Table I. Here absolute time is the time for reading all sectors and calculating hash values. We also check the pure reading time for reading all sectors of a hard disk, denoted as T_p. Then, we define the computation time as $T_a - T_p$. BTW, it takes 9,459 and 3,489 seconds for pure reading HD #1 and #2 respectively. As a comparison, results for the 3-$\mathcal{D}$ scheme are also given.

The data in Table I are consistent with the principle and analysis in the previous sections. Let us use HD #1 as an example. When efficiency of storage is considered, applying CGT with M_a can have a significant improvement that the storage needed (0.4MB) is about two orders of magnitude less than that of the 3-$\mathcal{D}$ scheme (29.8MB). Even with M_b, the number of stored hash values for CGT (2.5MB) is still tenfold less than that of the 3-$\mathcal{D}$ scheme, while both schemes' costs of hash calculations are similar (6,768 and 6,786 seconds respectively). When efficiency of time is considered, we found that applying CGT with M_c requires the least amount of computation (3,964 seconds).

We also perform some experiments to test the time of re-computation (T_r) for hash values affected by the operation of updating sectors, assuming that the updated sectors are in the same block. The experiments are run only for HD #1. As shown in Table II, more sparse the CGT matrix is, less time and calculations are needed for re-computation. When the number of updated sectors (# of updated sect.) is increased to 10, the number of calculations (# of cal.) for re-computation with M_a is almost the same as that for computing all sectors in the block. However, the number of calculations can be decreased to a relative lower level when M_b, is applied. Using M_c can further reduce the number of calculation since the

TABLE I
TIME AND STORAGE NEEDED FOR CGT SCHEME WITH M_a, M_b, M_c AND FOR 3-$\mathcal{D}$ SCHEME

		HD #1	HD #2
CGT(M_a)	T_a(Sec) / T_c(Sec) / S(MB)	20,488 / 11,029 / 0.4	7,431 / 3,942 / 0.1
CGT(M_b)	T_a(Sec) / T_c(Sec) / S(MB)	16,227 / 6,768 / 2.5	5,287 / 1,798 / 0.6
CGT(M_c)	T_a(Sec) / T_c(Sec) / S(MB)	13,423 / 3,964 / 25.1	4,704 / 1,215 / 6.1
3-$\mathcal{D}$	T_a(Sec) / T_c(Sec) / S(MB)	16,245 / 6,786 / 29.8	5,248 / 1,759 / 11.5

TABLE II
TIME NEEDED FOR RE-COMPUTATION WITH CGT SCHEME

# of updated sect.		T_r(sec)	# of cal.	# of hash
1	CGT(M_a) / CGT(M_b) / CGT(M_c)	363.92 / 18.33 / 7.61	1,176,456 / 29,703 / 1,324	8 / 3 / 2
10	CGT(M_a) / CGT(M_b) / CGT(M_c)	500.22 / 135.72 / 8.42	6,394,972 / 287,128 / 13,237	50 / 39 / 20
100	CGT(M_a) / CGT(M_b) / CGT(M_c)	502.09 / 475.17 / 66.30	8,000,000 / 1,861,386 / 126,407	51 / 188 / 191

number of hash values affected (# of hash) is decreased to 20. This preliminary experiment shows that it is better to apply M_c matrix to users who often modify some of the sectors and want to keep integrity evidence of the whole hard disk as well.

V. CONCLUSION

The hard disk integrity problem is one of the important problems in digital forensics investigation. This problem is solved by a seemingly unrelated technique, called combinatorial group testing. Then STD construction is adapted to verify the integrity of sectors in a hard disk to ensure the digital evidence stored in the sectors have not been modified even if some of the sectors may become bad sectors or be deleted legally during the investigation. It is shown to be used for normal users to keep track the integrity of their hard disks.

Current practice is to "freeze" the whole hard disk once the computer of a suspect has been seized. Future direction includes whether a quick preliminary investigation on the hard disk can identify the parts of hard disk to be frozen instead of freezing the whole hard disk since the volume of a hard disk is getting larger and larger, the hard disk integrity problem as well as the investigation become more and more difficult. Another future direction is to further improve our scheme to minimize the storage for hash values and reduce the computation time for computing the hash values. And how practical the scheme is for normal daily users to keep track of the integrity of their hard disks (e.g. to detect if any files have been modified by virus) also requires more detailed analysis and investigation.

ACKNOWLEDGMENT

The work described in this paper was partially supported by the General Research Fund from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. RGC GRF HKU 713009E), and HKU Seed Funding Programme for Basic Research Grant No. 200711159072.

REFERENCES

[1] Guide Software, EnCase (www.guidancesoftware.com).
[2] K. P. Chow, C. F. Chong, K. Y. Lai, L. C. K. Hui, K. H. Pun, W. W. Tsang and H. W. Chan, Digital Evidence Search Kit, *Proceedings of the First International Workshop on Systematic Approaches to Digital Forensic Engineering*, 2005.
[3] F.Y.W. Law and P. K.Y. Lai and Z. L. Jiang and R.S.C. Ieong and M. Y.K. Kwan and K.P. Chow and Lucas C.K. Hui and S.M. Yiu, Protecting digital legal professional privilege (LPP) data, *Proceedings of the 3rd International Workshop on Systematic Approaches to Digital Forensic Engineering (IEEE/SADFE-2008)*, pages 91-101, 2008.
[4] Z. L. Jiang, L. C. K. Hui, K. P. Chow, S. M. Yiu and P. K. Y. Lai. Improving Disk Sector Integrity Using 3-dimension Hashing Scheme, *Proceedings of The 2007 International Workshop on Forensics for Future Generation Communication*, volume 2, pages 141-145, 2007.
[5] Z. L. Jiang, L. C. K. Hui and S. M. Yiu. Analysis of K-Dimension Hashing Scheme to Improve Disk Sector Integrity, *IFIP International Federation for Information processing*, volume 285, pages 87-98, 2008.
[6] M. Goodrich, M. Atallah, and R. Tamassia. Indexing information for data forensics. *Applied Cryptography and Network Security Conference (ACNS)*, pages 206-221, 2005.
[7] J. Kornblum, Identifying almost identical files using context triggered piecewise hashing, *Proceedings of the 6th Annual Digital Forensic Research Workshop*, 2006.
[8] R. Dorfman, The detection of defective members of large populations, *Annals of Mathematical Statistics*, volume 14, pages 436-440, 1943.
[9] M. Sobel and P. A. Groll, Group testing to eliminate efficiently all defectives in a. binomial sample, *The Bell Systems Technical Journal*, volume 38, pages 1179-1253, 1959.
[10] D. Z. Du and F. K. Hwang, Combinatorial Group Testing and Its Applications, 2nd ed. Singapore World Scientific, 2000.
[11] C. J. Colbourn, J. H. Dinitz, and D. R. Stinson, Applications of combinatorial designs to communications, cryptography, and networking, *Surveys in Combinatorics*, volume 187, pages 37-100, 1993.
[12] D. J. Balding and D. C. Torney, Optimal Pooling Designs with Error Detection, *Journal of Combinatorial Theory, Series A*, volume 74, pages 131-140, 1996.
[13] A. J. Macula, A simple construction of d-disjunct matrices with certain constant weights, *Discrete Mathematics*, volume 162, pages 311-312, 1996.
[14] H. Q. Ngo and D.-Z. Du, A survey on combinatorial group testing algorithms with applications to DNA library screening, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 55, pages 171-182, 2000.
[15] N. Thierry-Mieg, A new pooling strategy for high-throughput screening: the Shifted Transversal Design, *Bmc Bioinformatics*, volume 7, page 13, 2006.
[16] N. Thierry-Mieg and G. Bailly, Interpool: interpreting smart-pooling results, *Bioinformatics*, volume 24, pages 696-703, 2008.