A Model of Survivable Storage System Based on Information Hiding

Qingjie Zhang, Jianming Zhu, and Yiheng Wang

School of Information, Central University of Finance and Economics,
Beijing, P. R. China 100081
cufe_dbzy@163.com, tyzjm65@163.com,
wangyiheng722@163.com

Abstract. A new model of survivable storage system based on the information hiding, which is called SSSBIH, is presented in this paper, This SSSBIH model is derived from the PASIS model. SSSBIH model can make stored data more security than PASIS model. We design the information hiding function in client agent and describe its principle of work in this paper. Note that data tampering from internal intruders is difficult to be detected nowadays. The information hiding function can detect tampering whatever any user accesses the data. With threshold themes, our model can carry out effective recovery for tampered data. Our model also doesn't need the history pool of the old data versions. This can save the storage space of storage nodes. At last, we give out an information hiding algorithm based on the discrete cosine transformation and make a simulation. In short, our model can enhance data credibility, survivability and security of a storage system.

Keywords: survivable storage system, information hiding, intrusion diagnosis and recovery, discrete cosine transformation.

1 Introduction

As the society increasingly relies on digitally stored and accessed information, supporting the availability, integrity, and confidentiality of information is crucial. The system should securely store users' critical information and ensure that the data is kept confidential and continuously accessible and cannot be destroyed. A *survivable storage system* can provide these guarantees.

Paul Stanton [1] compared confidentiality, data integrity, reliability and performance of the eight existing security storage systems, which are NASD, PASIS, S4, CFS, SFS-RO, SNAD, PLUTUS and SiRiUS. All of these systems share a common goal: to protect stored data from malicious adversaries. However, the design approaches to reach this goal vary tremendously in each system.

P. Sénac, M. Ott, and A. Seneviratne (Eds.): ICWCA 2011, LNICST 72, pp. 228–240, 2012.

PASIS is a storage system that encodes information via threshold schemes so as to distribute trust amongst storage nodes in the system [4]. The PASIS architecture combines decentralized storage system technologies, data redundancy and encoding, and dynamic self-maintenance to create survivable information storage. It strives to prevent data by storing elements of a file in different locations so that a single compromised server cannot disclose the entire relevant information. PASIS increases data availability in the face of failed servers [1].

By using the technology of information hiding, we present a new model named SSSBIH (Survivable Storage System Based on Information Hiding). In the client sides, an agent is added for the management capabilities of information hiding, and its main function is to detect attacks from internal intruders. Thus, it can improve security and survivability of a storage system.

In section 2, we present the architecture of SSSBIH. In section 3, we describe the service properties of our model. In section 4, we describe the algorithms used in our model. In section 5, we evaluate the performance of SSSBIH via simulations. In section 6, we describe the advantage of SSSBIH model. Finally, we conclude our paper and discuss future directions in Section 7.

2 SSSBIH

Survivable systems operate from the fundamental design thesis that no individual service, node, or user can be fully trusted. Survivable storage systems must distribute data among many nodes. Individual storage node must not expose information to anyone.

Threshold schemes, also known as secret sharing or information dispersal algorithms [1], offer a method that provides both information confidentiality and availability in a single, flexible mechanism. These schemes encode, replicate, and divide information into multiple pieces, or shares that can be stored at different storage nodes. The system can only reconstruct the original information when enough shares are available.

This section presents an overview of the PASIS system and the SSSBIH system, and the difference of them. More details of the PASIS system can be found in [4] and [5].

2.1 SSSBIH Architecture

The SSSBIH architecture, shown in Figure 1, combines decentralized storage systems, data redundancy and encoding, and dynamic self-maintenance to achieve survivable information storage. It is similar with the PASIS architecture. Most of decentralized storage systems are the similar architecture.

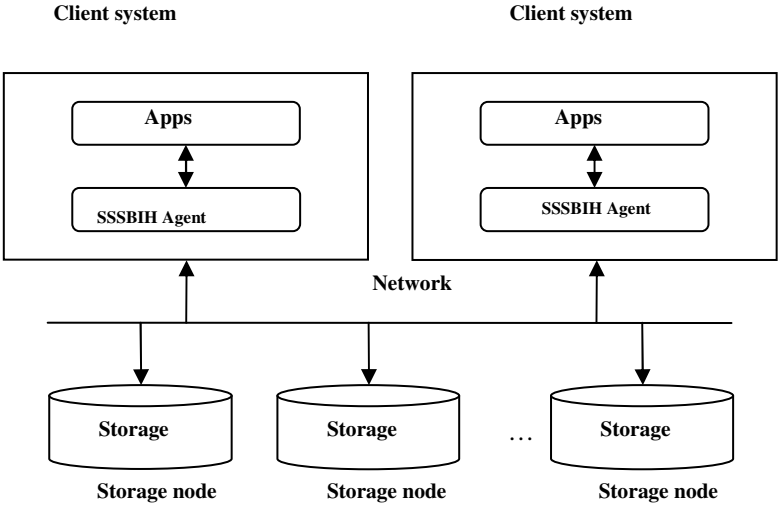

Fig. 1. SSSBIH architecture

2.2 SSSBIH System Components and Operation

Figure 3 presents the design of the SSSBIH Agent, which is derived from the PASIS Agent in figure 2.

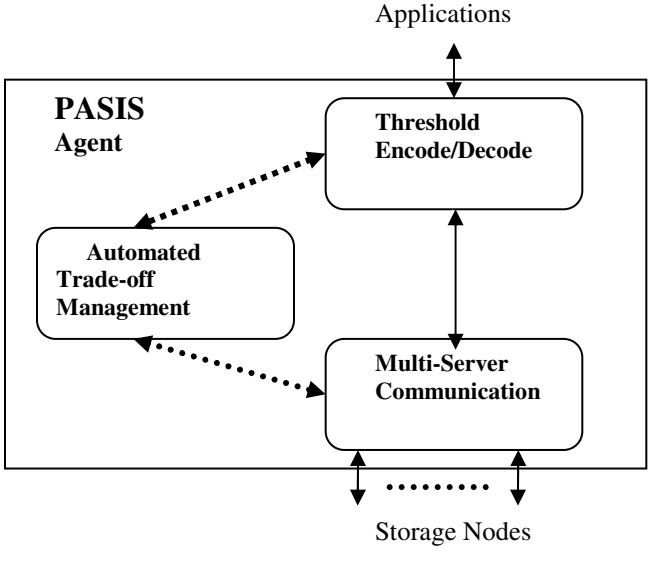

Fig. 2. PASIS agent

A SSSBIH system includes clients and servers. The servers, or storage nodes, provide persistent storage of shares; the clients provide all other aspects of SSSBIH functionality. Specifically, SSSBIH agents communicate with collections of SSSBIH servers to collect necessary shares and combine them using threshold schemes. A SSSBIH system uses threshold schemes to spread information across a decentralized collection of storage nodes. Client-side agents communicate with the collection of storage nodes to read and write information. The information hiding component is implemented in SSSBIH agents.

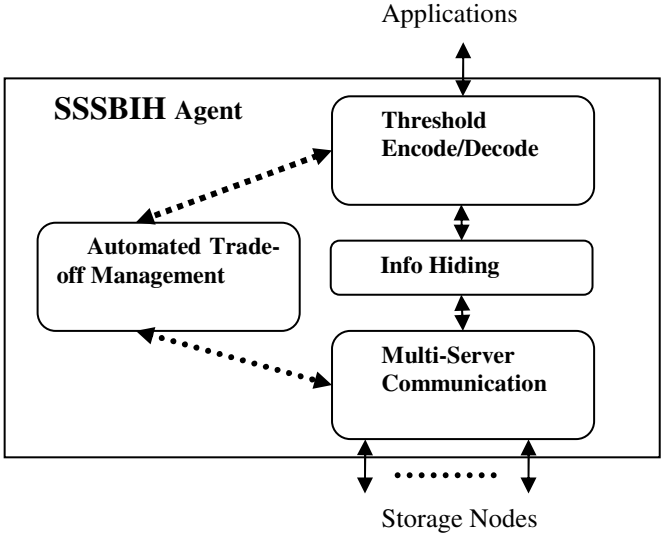

Fig. 3. SSSBIH agent

As with any distributed storage system, SSSBIH requires a mechanism that translates object names-for example, file names-to storage locations. A directory service shows the names of information objects stored in a SSSBIH system to the names of the shares. A share's name has two parts: the name of the storage node on which the share is located and the local name of the share on that storage node. A SSSBIH file system can embed the information needed for this translation in directory entries.

2.3 Automatic Trade-Off Management

For the SSSBIH architecture to be as effective as possible, it must make the full flexibility of threshold schemes available to clients. We believe this option requires automated selection of appropriate threshold schemes on a per-object basis. This selection should combine object characteristics and observations about the current system environment. For example, a SSSBIH client could use short secret sharing to store an object larger than a particular size, and conventional secret sharing to store

smaller objects. The size that determines which threshold scheme to use could be a function of the object type, current system performance, or both. As another example, an object marked as archival for which availability and integrity are the most important storage characteristics-should use an extra-large n. For read write objects, increased write overhead makes large n values less desirable. Moreover, if the archival object is also marked as public-such as a Web page-the client should ignore confidentiality guarantees when selecting the threshold scheme.

System performance observations can also be used to dynamically improve per-request performance. For example, clients can request shares from the m storage nodes that have responded most quickly to their recent requests. Storage nodes can also help clients make these decisions by providing load information or by asking them to look elsewhere when long response times are expected.

3 Service Properties

Secure storage servers sometimes face undesirable requests from legitimate user accounts. These requests can originate from malicious users, rogue programs e.g., e-mail viruses run by unsuspecting users, intruders exploiting compromised user accounts, or even normal legitimate user. Real users may abuse their access to data on the implementation of intentional or unintentional tampering.

In particular, they can modify or delete their accessible data. Even after an intrusion has been detected and terminated, system administrators still face two difficult tasks: determining the damage caused by the intrusion and restoring the system to the state before the intrusion. Especially, the restoration often requires a significant amount of time, reduces the availability of the system, and may cause data losses. SSSBIH offers a solution to these problems.

This section describes the problems of client-side intrusion diagnosis and recovery, and designs storage method based on information hiding.

3.1 Intrusion Diagnosis and Recovery

After gaining access to a system, an intruder has several ways to attack it. Most intruders attempt to destroy evidence of their presence by erasing or modifying system log files. Many intruders also install back doors in the system, allowing them to gain access at will in the future. They may also install malicious software, read and modify sensitive files, or use the system as a platform for launching additional attacks and so on. Once an intrusion has been detected and terminated, the system administrator is left with two difficult tasks: diagnosis and recovery[8].

Diagnosis is challenging because intruders sometimes can compromise the “administrator” account on most operating systems, giving them full control over all resources. In particular, they can manipulate audit logs, file modification times, and tamper detection utilities. Recovery is difficult. In this section we will discuss intrusion diagnosis and recovery in detail, and in the next section we will describe how SSSBIH deals with them.

3.1.1 Diagnosis

If an intruder tampers a data segment on a storage node, then the information hiding management component will discover that the data segment has encountered distortion, similarly, if the corresponding information hidden on the client agent is tampered, the information hiding management component will also discover it. But, its restoration is the lag, only when some data object accessed then to this object carries on the detection.

Intrusion diagnosis consists of three phases: firstly, take out hidden information from a data segment. Then compare the hidden information with the one preserved on the client-side agent. Finally, according to the comparison result, determines whether the data segment is tampered and credible.

If the data segment is credible, we enter the normal data access stage. When we verify that all data segments of a file are credible, we can combine them and form the whole file. If it is incredible, we enter the data recovery stage.

3.1.2 Recovery

Recovery aims at restoring incredible data. Recovery data come from the redundant data which are saved on some storage nodes. In general, we update the incredible data segment by its credible backup. At first we should check whether this backup is credible using the method in Section 3.1.1, and we do the following recovery if it is.

If the hidden information of the data segment's backup is identical to that preserved in the client agent, we update the original data segment by the backup. Otherwise, it indicates that the hidden information of the original data segment is tampered, and we update it by the hidden information of the backup. In this way, we enhance security and survivability of a storage system.

3.2 SSSBIH Design

SSSBIH have two advantages: it safeguards secure data storage by information hiding technology, and it realizes data recovery using threshold schemes.

The following is the detailed explanation on it.

3.2.1 SSSBIH Agent Info Hiding Component

Figure 4 shows the work flow of the tamper detection and data recovery of SSSBIH agent Info Hiding component.

Data pieces refer to data segments after being processed by threshold schemes.
Hiding Info refers to performing the information hiding operation.
Pieces in node refer to storing the data segments in the storage node.
Hidden info in c refers to storing the hidden information on the client.
Comp Hidden info refers to comparing the hidden information which is from the data segment with the one preserved on the client.
Credible pieces refer to reconstructing the data object with all of its credible data segments.
Data recovery refers to the process in Section 3.1.2

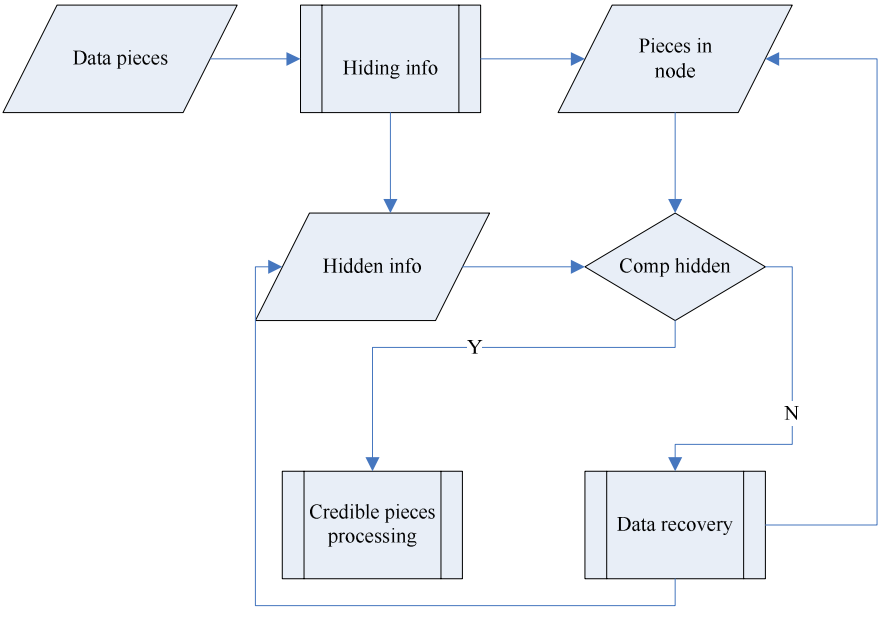

Fig. 4. SSSBIH agent Info Hiding elements

3.2.2 System Survivability

We assume a Byzantine fault model for servers, where a compromised node can behave arbitrarily.

In SSSBIH agent, Info Hiding component can enhance the survivability and the security of a storage system. This kind of safeguard stems from its mechanism to detect intrusion and to recover data. No matter an intruder modifies data segments or hidden information preserved in clients, the system can detect them and recover them by using redundant data.

This kind of recovery ability especially refers to the recovery of data segments distorted by malicious intruders.

If there is no hidden information in data segment, the above distortion is not easy to be detected.

3.3 Based on Information Hiding Storage Summary

Based on the information hiding function implemented in client agents, SSSBIH enhances survivability, security and credibility of a storage system.

For a data segment in many nodes redundancy storage with information hiding in it, can carry on the restoration effectively data. It can detect attacks to the data segment, and can carry on effective restoration to the attacked data segment.

Intrusion diagnosis consists of three phases: extract information which hidden in the original data, compare the hidden information with the one preserved on client agent, and determine whether the data segment is credible according to comparison result.

Incredible data segments can be recovered if its credible backups can be found.

The approach can solve the problem as the size and complexity of the data grows which history pool of old data versions method brings. To save the storage space in the storage node is the strongpoint. Another strongpoint is to omit the maintenance about using history the pool method to the old versions.

4 Algorithms

We denote the set of replicas by R and identify each replica using an integer in 0, 1 …(R-1). For simplicity, we assume |R|=3*f*+ 1, where *f* is the maximum number of replica that may be faulty. [7]

Algorithm 1: Write hidden information

Step1: divide the data object into |R| data segments using threshold schemes;

Step2: carry on the information hiding operation to each data segment, and hide a bit sequence $\boldsymbol{h}$ in the data segment ($\boldsymbol{h}$ can be time stamp, file name or serial number of the data segment);

Step3: save and manage the hidden information in client agent.

Algorithm 2: Read and compare hidden information

Step1: for each data segment of the data object, the hidden information $\boldsymbol{h}$ is extracted.

Step2: compare $\boldsymbol{h}$ with the corresponding hidden information $\boldsymbol{h'}$ preserved in client agent.

Step3: judge whether this data segment is credible based on the comparison result, i.e.,

If $\boldsymbol{h}$ and $\boldsymbol{h'}$ are identical
Then marking the data segment as credible
Else recover the data segment
End If
Reconstructing the data object by all of its credible data segments
End

Algorithm 3: Recover data

For the data segment which needs to be recovered,

Step1: get the backup of the data segment from other nodes

Step2: extract the hidden information ($\boldsymbol{h_1}$) from the backup

Step3: compare the hidden information $\boldsymbol{h_1}$ with the corresponding hidden information $\boldsymbol{h_1'}$ preserved in client agent

Step4: judge this data segment to be credible

If the hidden information $\boldsymbol{h_1}$ and the corresponding one in client agent are consistent

Then replace the data segment in the faulty node with the one that just read

Else If the hidden information h_1 and the corresponding one in client agent are inconsistent

Then If the hidden information h_1 is equal to h that the hidden information to be gotten before

Then replace corresponding hidden information h' that saved in client agent of h or h_1 **Else** read the corresponding data segment in next node. Go to step2

End If

Else with the backup to rewrite this data object

End If

End If

End

In the following, we give an algorithm using DCT (discrete cosine transformation) to extract hidden information from a data segment.

Algorithm 4: Extract hidden information

Write the hidden bit h_k, k=l… J (m)

Step1: Divide the data segment into 8*8*8 size (512-byte) blocks.

Step2: Observe the relations between the two selected DCT coefficients (e.g., (a12) k and (a21) k) of a block and the next hidden bit. When need to invert the bit in hidden bit sequence, make the bit sequence can implicitly saved in a client agent, and make the bit and the selected coefficient constitutes one kind of specific relations. Said in detail, to all hidden bit h_k, k=l… J (m), makes following operation:

If (h_k =l and $(a_{12})_k > (a_{21})_k$) or (h_k =0 and $(a_{12})_k < (a_{21})_k$, then the relations already satisfied, does not make any change to the bit of the bit sequence's. Otherwise, must invert the bit of the bit sequences. Generate a new bit sequence h'.

Step3: save the new bit sequence h' in client agent.

We can obtain the implicit hidden information by compares the two DCT coefficients in each block. Reading hidden information is similar to writing hidden information. And we omit the details here.

5 Simulation and Evaluation

In this section, we evaluate the performance and the capacity of SSSBIH via simulations, in comparison with the history pool of old data versions method [8].)

5.1 Simulation Settings

We carry out SSSBIH simulation experiments by the MALAB software, focusing on its effectiveness, performance, and the space size occupied.

5.1.1 Experimental Setup

Hardware Environment: CPU 2GHz AMD Athlon (tm) 64 Processor 3200 +, 512M memory, 64G hard drive.

Software Environment:
Operating System: Windows XP professional version 2002 service pack 2.
Simulation software: MALAB Version 7.4.0.287 (R2007a)
Experimental data: 6 data files, and file sizes are as follows:
43Kb, 142Kb, 1134Kb, 2243Kb, 3330Kb, 4442Kb.

5.1.2 Experimental Content

a. We divide each file into three data segments and write hidden information in it.

b. We verify the credibility of data segments of a file according to Section 3.1, and assemble them.

c. We deliberately tamper data segments or the hidden information preserved at client agents for testing the system detection and recovery functions.

5.2 Evaluation

5.2.1 Validity

The experiments prove that SSSBIH model is valid. First, the simulations show the system can successfully divide a file and write hidden information. Secondly, Assembling of data segments can be completed successfully. Last but not the least, when the data segments stored on storage nodes or the hidden information stored on client agents are tampered, the system can successfully detect them.

5.2.2 Performance

Table 1 shows the file size and the processing time of the experiments, including the time of file division and the time of file assembly.

Table 1. Experimental files size and processing time (seconds)

Experiment number	File size	The time of file division	The time of file assembling
1	43Kb	0.0887	0.1017
2	142Kb	0.1286	0.1190
3	1134Kb	0.8626	0.7386
4	2243Kb	1.5177	1.4028
5	3330Kb	2.3773	2.0603
6	4442Kb	2.8417	2.7149

Figure 5 shows visual display the relationships of the file size and processing time. It can be seen from the figure: split file processing time is slightly higher than the combination of the file, and between the file size and processing time are the basic linear.

As the hardware configuration increases and the program to be compiled, the model fully meets performance requirements.

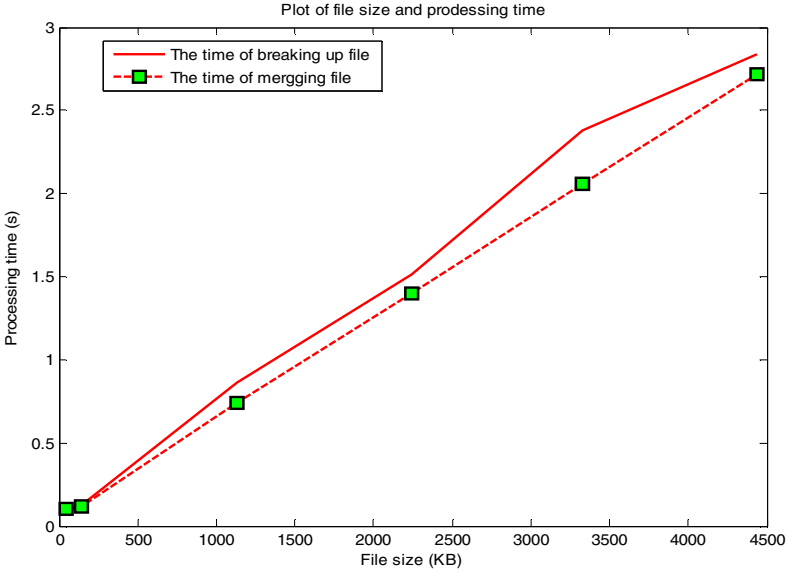

Fig. 5. SSSBIH Performance

5.2.3 Capacity

In comparison with the history pool method, SSSBIH can greatly save storage space because it does not allocate space for old-version data and it only use a small amount of space for hidden information at client agents. The storage space of hidden information is not more than 1/4096 of that of the data.

The size of hidden information is:

Hsize = (data segment size) / 4096. This is because according to Algorithm 4 each block (512 bytes) corresponds to one bit of hidden information.

Applying history pool method, a 10GB history pool can provide a detection window of between 50 and 470 days [8].

However, our method does not need to save historical data, and can save 20% of storage space in each storage node at least.

Therefore, if the amount of data stored does not exceed disk space limit, the detection window that our method provides, is unlimited.

6 The Advantage of SSSBIH Model

6.1 To Compare with PASIS Model

In PASIS model, even if operating systems and communication are totally normal, intruders also can tamper data segments without affecting the data reconstruction. This is one of faults of PASIS model. Data reconstructed by tampered segments will send wrong information. And wrong information often causes significant loss.

SSSBIH model can solve this problem with the combination of threshold schemes and information hiding. Using information hiding technology, we can detect data segments tampering. No matter invaders modify data in storage nodes or hidden information in client agent, the SSSBIH system can discover tampering with data.

In addition, hidden information implicitly stored in client, can effectively reduce the possibility that intruders maliciously tamper with hidden information and matched data segments at the same time. This also improves system's survivability.

6.2 To Compare with S4: Self-Securing Storage System

S4: self-securing storage system model adopts the history pool of old data versions method. It monitors tampering with data then uses the old data to recover the tampered data.

The history pool of old data versions method of S4 will produce large amounts of redundant data. And a lot of storage space will be occupied. However, the more amount of data increase, the more complex the maintenance for the old version is. That will make the system easily be intruded.

By using the information hiding method of SSSBIH model, we can prompt detect tampering with data. So the maintenance of the old versions will be eliminated, and storage nodes of storage space will be saved. Thus, it improves the security and survivability of system. On the other hand, S4 model's recovery is lagging behind the tamper with the data was found, while perhaps tampered data has been read by some users. In SSSBIH model, at the first stage of users' accessing, tampered data is found and recovered. So it can guarantee the read data is correct.

7 Summary

This paper proposes a survival storage system model based on the unification of threshold schemes and information hiding. It describes the client agent information hiding part's composition and the principle of work in detail and gives an information hiding algorithm based on the discrete cosine transformation for credible storage system.

This model has the merits of detecting tampering from internal intruders and saving storage space.

This paper mentioned the data storage can save more space of a storage node than the Self-securing storage. In this paper, we only present non-compression data storage and we will study compressed data storage in the future.

Acknowledgment. This research is supported by National Natural Science Foundation of China (Grant No. 60573035, 60743005).

This research is supported by The National Natural Science Foundation of China (No.60970143)

This research is also supported by the Key Project of Chinese Ministry of Education. (No.109016)

This research is supported by the third stage of "Project 211" of Central University of Finance and Economics.

References

1. Stanton, P.: Securing Data in Storage: A Review of Current Research, http://www.projects.ncassr.org/storage-ec/papers/stantontechnicalreport2004.pdf
2. Chockler, G., Lynch, N.: Fault-Tolerant Distributed Storage. MIT Computer Science and Artificial Intelligence Laboratory (2004)
3. Suhail, M.A., Obaidat, M.S.: Digital Watermarking-Based DCT and JPEG Model. IEEE Transactions on Instrumentation and Measurement 52(5) (October 2003)
4. Gregory, R., et al.: Survivable Storage Systems. In: DARPA Information Survivability Conference and Exposition, Anaheim, CA, June 12-14, vol. 2, pp. 184–195. IEEE, Los Alamitos (2001)
5. Wylie, J.J., Bigrigg, M.W., Strunk, J.D., Ganger, G.R., Kılıççöte, H., Khosla, P.K.: Information Storage Systems, Computer, Carnegie Mellon University (August 2000)
6. Hayashi, D., Miyamoto, T., Doi, S., Kumagai, S.: Agents for Autonomous Distributed Secret Sharing Storage System. In: Proc. 2002 International Conference on Circuit/Systems Computers and Communications (2002), http://www.kmutt.ac.th/itc2002/CD/pdf/17_07_45/WP1_PJ/6.pdf
7. Goodson, G.R., Wylie, J.J., Ganger, G.R., Reiter, M.K.: Efficient Byzantine-tolerant erasure-coded storage. Appears in the Proceedings of the International Conference on Dependable Systems and Networks, Carnegie Mellon University (June 2004)
8. Strunk, J.D., Goodson, G.R., Scheinholtz, M.L., Soules, C.A.N., Ganger, G.R.: Self-Securing Storage:Protecting Data in Compromised Systems. In: Proceedings of the Foundations of Intrusion Tolerant Systems (OASIS 2003), Carnegie Mellon University. IEEE, Los Alamitos (2003)
9. Maheshwari, U., Vingralek, R., Shapiro, W.: How to Build a Trusted Database System on Untrusted Storage, http://www.usenix.org/events/osdi00/full_papers/maheshwari/maheshwari.pdf
10. Riedel, E., Kallahalla, M., Swaminathan, R.: A framework for evaluating storage system security. Appears in the Proceedings of the 1st Conference on File and Storage Technologies (FAST), Hewlett-Packard Laboratories, Palo Alto, California, Monterey, CA (January 2002)
11. Xu, L.: Hydra: A Platform for Survivable and Secure Data Storage Systems. In: Proceedings of the 2005 ACM workshop on Storage security and survivability, Virginia, USA, November 11 (2005)
12. Lakshmanan, S., Ahamad, M., Venkateswaran, H.: Responsive Security for Stored Data. IEEE Transactions On Parallel And Distributed Systems 14(9) (September 2003)
13. Zhu, J., Wang, C., Ma, J.: Intrusion-Tolerant Based Survivable Model of Database System. Chinese Journal of Electronics 14(3) (July 2005)