SeDas: A Self-destructing Data System based on Active Storage Framework

Lingfang Zeng, *Member*, *IEEE*, Shibin Chen, Qingsong Wei, *Member*, *IEEE*, Dan Feng, *Member*, *IEEE*

Personal data stored in the Cloud may contain account numbers, passwords, notes, and other important information that could be used and misused by a miscreant, a competitor, or a court of law. These data are cached, copied, and archived by Cloud Service Providers (CSPs), often without users' authorization and control. Self-destructing data mainly aims at protecting the user data's privacy. All the data and their copies become destructed or unreadable after a user-specified time, without any user intervention. Besides, the decryption key is destructed after the user-specified time. In this paper, we present *SeDas*, a system that meets this challenge through a novel integration of cryptographic techniques with active storage techniques based on T10 OSD standard. We implemented a proof-of-concept *SeDas* prototype. Through functionality and security properties evaluation of the *SeDas* prototype, the results demonstrate that *SeDas* is practical to use and meets all the privacy-preserving goals described above. Compared with the system without self-destructing data mechanism, throughput for uploading and downloading with the proposed SeDas acceptably decreases by less than 72%, while latency for upload/download operations with self-destructing data mechanism increases by less than 60%.

***Index Terms*—Self-destructing data, active storage, object-based storage, Cloud computing.**

I. Introduction

With development of Cloud computing and popularization of mobile Internet, Cloud services become more and more important in people's life. People are more or less requested to submit or post some personal private information to Cloud by Internet. When people do this, they subjectively hope service providers will provide security policy to protect their data from leaking to the other people.

As people rely more and more on the Internet and Cloud technology, security of their privacy takes more and more risk. On the one hand, when data is being processed, transformed and stored by current computer system or network, systems or network must cache, copy or archive it because these copies are essential for systems and network. However, people have no knowledge about these copies and cannot control them, thus leaking the privacy of these copies to the public. On the other hand, their privacy also can be leaked attributing to CSPs' (Cloud Service Providers) negligence, hackers' intrusion or some legal actions. These problems present formidable challenges to protect people's privacy in the Cloud environment.

Pioneering study of Vanish [1] supplies a new idea for sharing and protecting privacy. In the Vanish system, secret key is divided and stored in a P2P system with Distributed Hash Tables (DHTs). With joining and exiting of P2P node, the system can maintain secret keys. According to characteristics of P2P, after about eight hours the DHT will refresh each node. With Shamir Secret Sharing Algorithm [2], when one cannot get enough parts of a key, the encrypted data cannot be decrypt with the key which indicates the key is destroyed.

Except some special attacks to characteristics of P2P are challenges of Vanish [3], [4], the uncontrollable surviving time of the key is also one of the disadvantages for Vanish.

Aiming at these disadvantages, this paper proposes a solution to implement a self-destructing data system, or *SeDas*, which is based on active storage framework [5], [6], [7], [8], [9], [10].

The SeDas system defines two new modules, a self-destruct method object that is associated with each secret key part and survival time parameter for each secret key part. In this case, SeDas can meet the requirements of self-destructing data with controllable survival time while users can use this system as a general object storage system.

Our contributions are summarized as follows:

(1) We focus on the related key distribution algorithm, Shamir's algorithm [2], which is used as the core algorithm to implement client (users) distributing key in object storage system. We use these methods to implement safety destruct with equal divided key (Shamir secret shares [2]).

(2) Based on active storage framework, we use object-based storage interface to store and manage the equal divided key. We implemented a proof-of-concept SeDas prototype.

(3) Through functionality and security properties evaluation of SeDas prototype, the results demonstrate that SeDas is practical to use and meets all the privacy-preserving goals. The prototype system imposes reasonably low runtime overhead.

(4) SeDas supports security erasing files and random encryption keys stored in Hard Disk Drive (HDD) or Solid State Drive (SSD) respectively.

The rest of this paper is organized as follows. We review the related work in Section II. We describe the architecture, design and implementation of SeDas in Section III. The extensive evaluations are presented in Section IV. And we conclude this paper in Section V.

This work was supported in part by the National 973 Program of China under Grant No. 2011CB302301, China National Funds for Distinguished Young Scientists under Grant No. 61025008, and A-STAR, Singapore under Grant No. 112 172 0010.

Lingfang Zeng, Dan Feng are with Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, China, 430074 (E-mail:{lfzeng,dfeng}hust.edu.cn).

Qingsong Wei and Shibin Cheng with Data Storage Institute, A*STAR, 138632, Singapore (E-mail: {WEI_Qingsong,Chen_Shibin@dsi.a-star.edu.sg).

II. RELATED WORK

A. Data Self-Destruct

The self-destructing data system in the Cloud environment should meet the following requirements: (1) How to destruct all copies of the data simultaneously and make them unreadable in case of the data is out of control? Local data destruction approach will not work in the Cloud storage because the number of the backups or archives of the data that is stored in the Cloud are unknown, and some nodes preserving the backup data have been offline; The clear data should become permanently unreadable because of the loss of encryption key, even if an attacker can retroactively obtain a pristine copy of that data; (2) Without any explicit delete actions by the user, the parties storing that data or any un-trusted third party; (3) Without the need to modify any of the stored or archived copies of that data; (4) Without the use of secure hardware but supporting to completely erase data in HDD and SSD respectively; (5) No new external services need to be deployed.

Tang et al. [11] proposed FADE which is built upon standard cryptographic techniques and assuredly deletes files to make them unrecoverable to anyone upon revocations of file access policies. Wang et al. [12] utilized the public key based on homomorphism authenticator with random mask technique to achieve a privacy-preserving public auditing system for Cloud data storage security, and uses the technique of bilinear aggregate signature to handle multiple auditing tasks. Perlman et al. [13] presented three types of assured delete: expiration time known at file creation, on demand deletion of individual files, and custom keys for classes of data.

Vanish [1] is a system for creating messages that automatically self-destruct after a period of time. It integrates the cryptographic techniques with global-scale, P2P, and distributed hash tables (DHTs): DHTs discard data older than a certain age. The key is permanently lost, and the encrypted data is permanently unreadable after data expiration. Vanish works by encrypting each message with a random key and storing shares of the key in a large, public DHT. However, Sybil attacks [3] may compromise the Vanish system by continuously crawling the DHT and saving each stored value before it ages out and the total cost is two orders of magnitude less than that of mentioned in reference [14] estimated. They can efficiently recover keys for more than 99% of Vanish messages. Wolchok et al. [3] concludes that public DHTs like VuzeDHT [15] probably cannot provide strong security for Vanish. So, Geambasu et al. [14] proposes two main countermeasures.

Although, using both OpenDHT [16] and VuzeDHT might raise the bar for an attacker, at best it can provide the maximum security derived from either system: if both DHTs are insecure, then the hybrid will also be insecure. OpenDHT is controlled by a single maintainer that essentially acts as a trusted third party in this arrangement. It is also susceptible to attackers on roughly two hundred PlanetLab [17] nodes on which it runs, most of which are housed low-security research facilities. Though Vanish is a promising approach to an important privacy problem, it is insecure in its current form [3].

To address the problem of Vanish discussed above, in our previous work [4], we proposed a new scheme, called SafeVanish, to prevent hopping attack, which is one kind of the Sybil attacks [18][19], by extending the length range of the key shares to increase the attack cost substantially, and proposed some optimizations on the Shamir Secret Sharing algorithm [20] implemented in the Vanish system. Also, we presented an improved approach against sniffing attacks by using the public key cryptosystem to prevent data from sniffing operations.

However, the use of P2P features is still the fatal weakness both for Vanish and SafeVanish, because specific attacks against P2P methods (e.g. hopping attacks and Sybil attacks [3]) are still possible.

In addition, for the Vanish system, the survival time of key attainment is determined by DHT system and uncontrollable for users. Based on active storage framework, this paper proposes a distributed object-based storage system with self-destructing data function. Our system combines proactive approach in the object storage techniques and method object by using the powerful data processing capabilities inside OSDs to handle data self-destruction. User can specify the survival time of the distribution key and use the settings of expanded interface to export the life cycle of a key, allowing the user to control the subjective life-cycle of private data.

B. Object-based storage

Object-based Storage (OBS) [21] uses Object-based Storage Device (OSD) [22] as the underlying storage device. The T10 OSD standard [22] is being developed by the Storage Networking Industry Association (SNIA) and the INCITS T10 Technical Committee. Each OSD consists of CPU, network interface, ROM, RAM, and storage device (i.e. disk or RAID sub-system), and exports a high-level data object abstraction on the top of block device read/write interface.

With the emergence of object-based interface, storage devices can take advantage of the expressive interface to achieve some intelligent collaborations between the application servers and the storage devices. A storage object can be a file consisting of a set of ordered logical data blocks, or a database containing many files, or just a single application record such as a database record of one transaction. Information of the data is also stored as objects, which can include the requirements of Quality of Service (QoS) [23], security [24], caching, and backup. Kang et al. [28] even implemented the object-based model enables SCM (Storage Class Memories) devices to overcome the disadvantages of the current interfaces and provided new features such as object-level reliability and compression. In the last ten years, many systems, such as Lustre [25], Panasas [26] and Ceph [27] using object-based technology have been developed and deployed.

C. Active storage

Since the data can be processed in storage device, people attempt to add more functions into storage device (e.g. OSD) to make it more intelligent, called “Intelligent Storage” or

"Active Storage" [5], [6], [7], [8], [9], [10]. For instance, IDISK[29] and SmAS Disk[30] can offload application codes to disks, but the disks respond to I/O requests of clients passively. A stream-based programming model has been proposed for Active Disk [31], [32], [33], but the stream is allowed to pass through only one disklet (user-specific code).

Today, the active storage system has become one of the most important research branches in the domain of intelligent storage systems. For instance, Wickremesinghe et al. [34] proposed a model of load-managed active storage, which strives to integrate computation with storage access in a way that the system can predict the effects of offloading computation to ASUs (Active Storage Units). Hence, applications can be configured to match hardware capabilities and load conditions. MVSS [35], a storage system for active storage devices, provided a single framework to support various services at device level. MVSS separates the deployment of services from file systems, and thus allowing services to be migrated to the storage devices.

There have been several efforts to integrate active storage technology into the T10 OSD standard. References [5], [7], [8], [10] all proposed their own implementation of active storage framework for the T10 OSD standard. However, these implementations either are preliminary, or validate their systems on a variety of data intensive applications and fully demonstrate the advantage of object-based technology. Our work extends prior research (such as, Qin et al.'s [5], John et al.'s [7], Devulapalli et al.'s [8] and Xie et al.'s [10]) by considering data self-destruction in this area.

D. Completely erase bits of the encryption key

In SeDas, erasing files that include bits (Shamir secret shares [2]) of the encryption key, is not enough when we erase/delete a file from their storage media. The erased files can still be recovered until the erased areas on the disk are overwritten by new information. With flash-based Solid State Drives (SSDs), the erased file situation is even more complex due to the fact that SSDs have a very different internal architecture [36].

(1) For HDD: Several techniques that reliably delete data from hard disks are available as built-in ATA or SCSI commands, software tools (such as, DataWipe [37], HDDerase [38] SDelete [39]), and government standards (e.g. [40]). These techniques provide effective means of sanitizing HDDs: either individual files stored or the entire drive. Software methods typically involve overwriting all or part of the drive multiple times with patterns specifically designed to obscure any remnant data. For instance, different from erasing files simply marks file space as available for reuse, data wiping overwrites all data space on a storage device, replacing useful data with garbage data. Depending upon the method used, the overwrite data could be zeros (also known as "zero-fill") or could be various random patterns [41]. The ATA and SCSI command sets include "secure erase" commands that should sanitize an entire disk. Physical destruction and degaussing are also effective.

(2) For SSD: SSDs work differently from platter-based HDDs, especially when it comes to read and write processes on the drive. The most effective way to securely delete platter-based HDDs (overwriting space with data) becomes unusable on SSDs because of their different internal design. Data on platter based hard disks can be deleted by overwriting it. This ensures that the data is not recoverable by data recovery tools. This method is not working on SSDs because SSDs are different from HDDs in both the technology they use to store data and the algorithms they use to manage and access that data.

SSDs maintain a layer of indirection between the logical block addresses that computer systems use to access data and the raw flash addresses that identify physical storage.

The layer of indirection enhances the performance and reliability of SSDs by hiding flash memory idiosyncratic interface and managing its limited lifetime, but it can also produce copies of the data that are invisible to the user which a sophisticated attacker can recover. There are three different main techniques for sanitizing an entire SSD: issuing a built in sanitize command, repeatedly writing over the drive using normal I/O operations, and degaussing the drive [36], [41].

Analog sanitization is more complex for SSDs than for hard drives as well. Gutmann [42], [43] examines the problem of data remnants in flash, DRAM, SRAM, and EEPROM, and recently, so-called "cold boot" attacks [44] recovered data from powered-down DRAM devices. The analysis in [36] suggests that verifying analog sanitization in memories is challenging because there are many mechanisms that can imprint remnant data on the devices.Wei et al. [36] found that built-in commands are effective for SSDs, but manufacturers sometimes implement them incorrectly. For example, overwriting the entire visible address space of an SSD twice is usually, but not always sufficient to sanitize the drive. Their studies reveal that none of the existing hard drive-oriented techniques for individual file sanitization are effective on SSDs.

To the best of our knowledge, most of previous work aimed at some special applications, e.g., database, multimedia, etc., and there is no general system level self-destructing data in the literature. In order to substantiate our proposed SeDas, we have implemented a fully functional prototype system. Based on this prototype, we carry out a series of experiments to examine the functions of SeDas. Extensive experiments show that the proposed SeDas does not affect the normal use of storage system, and can meet the requirements of self-destructing data under a survival time of user controllable key.

III. DESIGN AND IMPLEMENTATION OF SEDAS

A. The SeDas architecture

Fig.1 shows the architecture of SeDas. There are three parties based on the active storage framework.

(1) **Metadata server**: The Metadata Server (MDS) is responsible for user management, server management, session management and file metadata management.

(2) **Application**: The application node is a client to use

storage service of the SeDas.

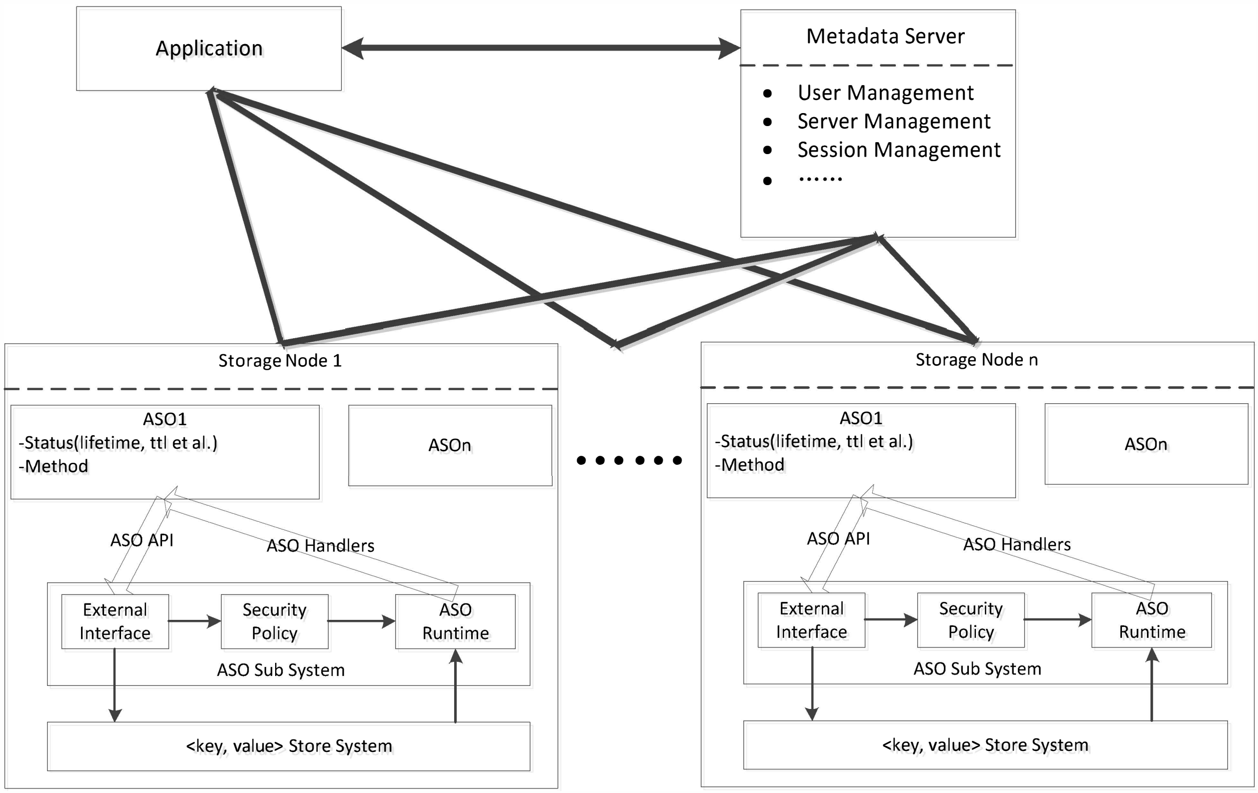


Fig. 1. The SeDas system architecture.

(3) **Storage node**: Each storage node is an OSD. It contains two core subsystems: <key,value> store subsystem and active storage object (ASO) runtime subsystem. The <key,value> store subsystem that is based on the object storage component is used for managing object stored in storage node: lookup object, read/write object and so on. The object ID is used as a key. The associated data and attribute are stored as values.

The ASO runtime subsystem based on the active storage agent module in the object-based storage system is used to process active storage request from users, manage method objects and policy objects.

B. Active storage object

An active storage object derives from a user object and has a time-to-live (ttl) value property. The *ttl* value is used to trigger the self-destruct operation. The *tll* value of a user object is infinite so that a user object will not be deleted until user deletes it manually. The *ttl* value of an active storage object is limited so an active object will be deleted when the value of the associated policy object is true.

Interfaces extended by *ActiveStorageObject* class are used to manage *ttl* value. The create member function needs another argument for *ttl*. If the argument is -1, *UserObject::create* will be called to create a user object, else, *ActiveStorageObject::create* will call *UserObject::create* first and associate it with the self-destruct method object and a self-destruct policy object with the *ttl* value. The *getTTL* member function is based on the *read_attr* function and returns the *ttl* value of the active storage object. The *setTTL*, *addTiime* and *decTime* member function is based on the *write_attr* function and can be used to modify the *ttl* value.

C. Self-destruct method object

Generally, kernel code can be executed efficiently; while, a service method should be implemented in user space with these following considerations.

Many libraries such as *libc*, can be used by code in user space but not in kernel space. Mature tools can be used to develop software in user space. It is much safer to debug code in user space than in kernel space.

A *service method* needs long time to process complicated task, so implementing code of a service method in user space can make advantage for performance of the system. The system might crash with an error in kernel code, but this will not happen if the error occurs in code of user space.

A *self-destruct method object* is a service method. It needs three arguments. The *lun* argument specifies the device, the *pid* argument specifies the partition and the *obj_id* argument specifies the object to be destructed.

D. Data process

To use SeDas system, user's applications should implement logic of data process, act as a client node. There are two different logics: uploading and downloading.

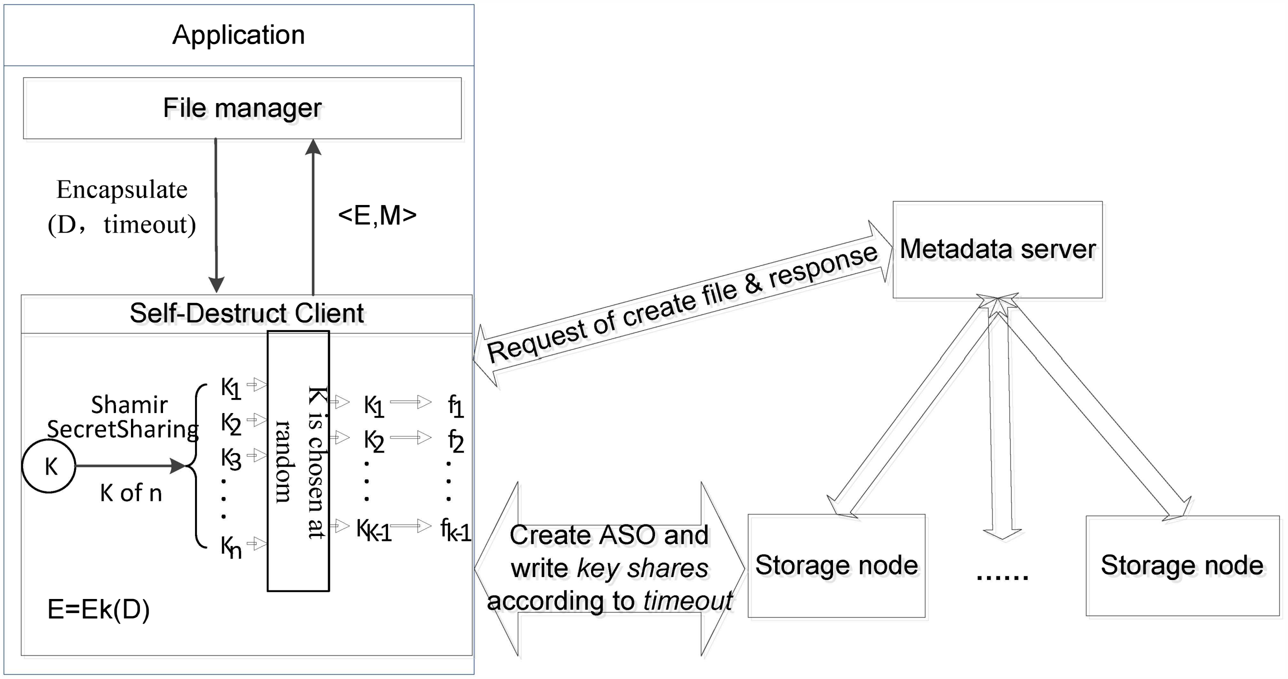
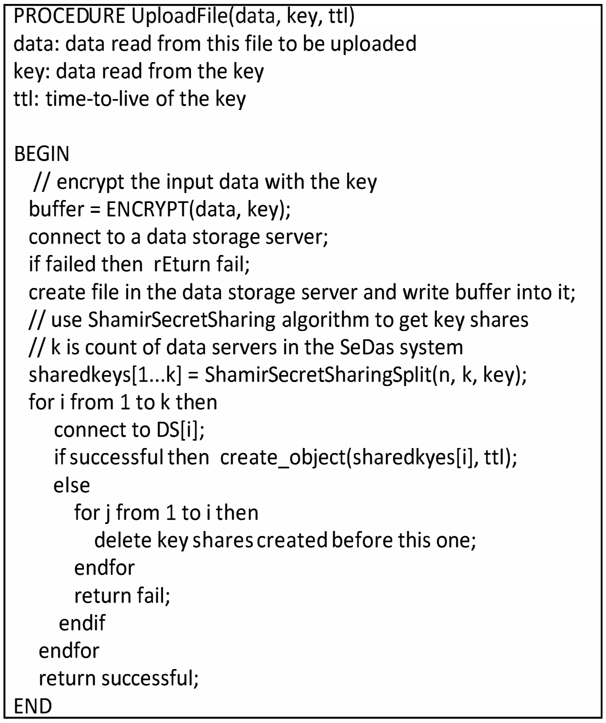


Fig. 2. Uploading file process.



```
PROCEDURE UploadFile(data, key, ttl)
data: data read from this file to be uploaded
key: data read from the key
ttl: time-to-live of the key

BEGIN
   // encrypt the input data with the key
  buffer = ENCRYPT(data, key);
  connect to a data storage server;
  if failed then  rEturn fail;
  create file in the data storage server and write buffer into it;
  // use ShamirSecretSharing algorithm to get key shares
  // k is count of data servers in the SeDas system
  sharedkeys[1...k] = ShamirSecretSharingSplit(n, k, key);
  for i from 1 to k then
      connect to DS[i];
      if successful then  create_object(sharedkyes[i], ttl);
      else
         for j from 1 to i then
            delete key shares created before this one;
         endfor
         return fail;
       endif
   endfor
   return successful;
END
```

Fig. 3. Uploading file.

(1) Uploading file process: (see Fig.2) When a user uploads a file to a storage system and store his key in this SeDas system, he should specify the file, the key and ttl as arguments for the uploading procedure. Fig.3 presents its pseudo code.

In these codes, we assume data and key has been read from the file. The ENCRYPT procedure uses common encrypt algorithm or user defined encrypt algorithm. After uploading data to storage server, key shares generated by ShamirSecretSharing algorithm will be used to create active storage object (ASO) in storage node in the SeDas system.

(2) Downloading file process: Any user who has relevant permission can downloads data store in the data storage system. The data must be decrypted before use. The whole logic is implemented in code of user's application.

In above code, we assume encrypted data and meta information of key has been read from the downloaded file. Before decrypting, client should try to get key shares from storage nodes in the SeDas system. If the self-destruct operation hasn't been triggered, the client can get enough key shares to reconstruct the key successfully. If the associated ASO of the key has been destructed, the client cannot reconstruct the key so he only read encrypted data.

E. Data security erasing in disk

We must secure delete sensitive data and reduce the negative impact of OSD performance due to deleting operation.

The proportion of required secure deletion of all the files is not great, so if this part of the file update operation changes, then the OSD performance will be impacted greatly.

Our implementation method as follows:

(1) The system pre-specifies a directory in a special area to store sensitive files.

(2) Monitor the file allocation table, and acquire and maintain a list of all sensitive documents, the logical block address (LBA).

(3) LBA list of sensitive documents appear to increase or decrease, the update is sent to the OSD.

(4) OSD internal synchronization maintains the list of LBA,

the LBA data in the list updates. E.g. for SSD, the old data page write 0, and then another to write the new data page. When the LBA list is shorter than the corresponding file size is shrinking. At this time, the old data needs to be corresponding to the page all write.

(5) For ordinary LBA, the system uses the regular update method.

(6) By calling of ordinary data erasure API, we can safely delete sensitive files of the specified directory.

Our strategy only changes a few sensitive documents the update operation, no effect on the operational performance of ordinary file. In general, the secure delete function is implied while the OSD read and write performance can be negligible.

IV. EVALUATION AND DISCUSSION

In this section we discuss test method and implementation for SeDas, and then give analysis on the test result. We put up a data storage file system based on pNFS in virtual machine environment to implement test for file uploading, downloading and sharing.

A. Experimental setup and methodology

There are multiple storage services for user to store data. Meanwhile, to avoid the problem produced by the centralized "trusted" third party, the responsibility of SeDas is to protect user key and provide the function of self-destructing data. Fig.4 shows the brief structure of user application program realizing storage process. In this structure, user application node contains two system clients: any third-party data storage system (TPDSS), and SeDas. User application program interact with SeDas sever through SeDas' client, getting data storage service.

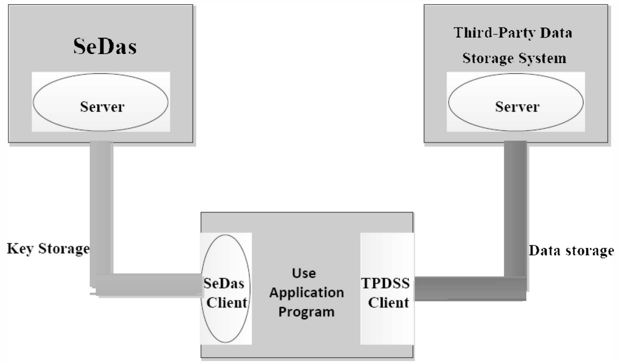


Fig. 4. Structure of user application program realizing storage process.

The way to attain storage service by client interacting with server depends on the design of TPDSS. We do not need a secondary development for different TPDSS. The process to store data has no change, but encryption is needed before uploading data and so is the decryption needed after downloading data. In the process of encryption and decryption, user application program interacts with SeDas. To test the implementation of SeDas described in above section, we use pNFS to put up a TPDSS to implement data storage service. The client mainly runs in kernel mode, and we can mount remote file system to local.

A VMware virtual environment is built up to test. The configuration of host and virtual node are as shown in Table.1.

Table. 1. Configuration of host and virtual node.

Node / System	Host	Virtual node
SeDas	CPU: Intel Pentium® Dual-Core CPU E6500 2.93GHz RAM: 2GB DDR2-800 SDRAM NIC: Realtek PCIe GBE Family Controller OS: Microsoft Windows 7 Ultimate 6.1.7601 HDD: WDC WD5000AADS-00S9B0 ATA Device 500GB	CPU: Intel Pentium® Dual-Core CPU E6500 2.93GHz RAM: 256MB VMware vRAM NIC: VMware NAT connecting virtual NIC OS: Fedora 12 Kernel: Linux 2.6.35 HDD: VMware SCSI virtual disk 30GB
pNFS	CPU: Intel Pentium® Dual-Core CPU E6600 3.0GHz RAM: 2GB DDR2-800 SDRAM NIC: Realtek PCIe GBE Family Controller OS: Microsoft Windows 7 Ultimate 6.1.7601 HDD: WDC WD5000AADS-00S9B0 ATA Device 500GB	CPU: Intel Pentium® Dual-Core CPU E6500 2.93GHz RAM: 256MB VMware vRAM NIC: VMware NAT connecting virtual NIC OS: Fedora 12 Kernel: Linux 2.6.35 HDD: VMware SCSI virtual disk 30GB

To avoid creating virtual machines repeatedly, we make the same configuration on every node. From a performance point of view, some adjustments may be needed, such as improving CPU configuration of metadata sever, increasing the size of the disk and memory for storage nodes. VMware version is VMware Workstation 7:1:3 build-324285.

B. Evaluation

The evaluation platform built up on pNFS supports simple file management, which includes some data process functions, such as, file uploading, downloading and sharing.

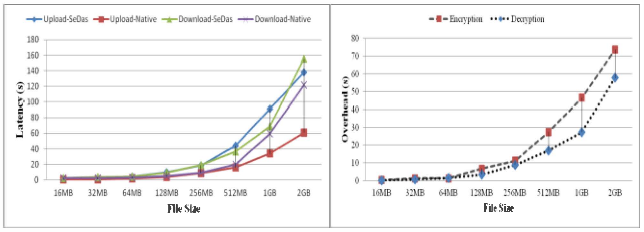
(1) Functional testing

We input the full path of file, key file, and the life time for key parts. The system encrypts data and uploads encrypted data. The life time of key parts is 150 seconds for a sample text file with 101 bytes. System prompts creating active object successful afterwards and that means uploading file gets completed. The time output finally is the time to create active object. SeDas was checked and corresponded changes on work directory of storage node. The sample text file also was downloaded or shared successfully before key destruct.

(2) Performance evaluation

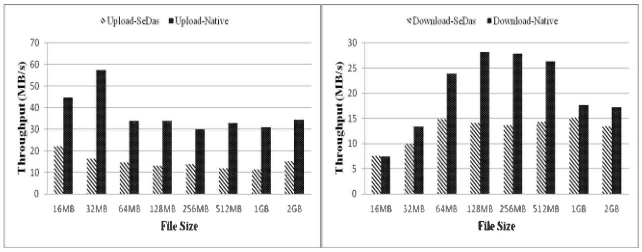
As mentioned above, the difference of I/O process between SeDas and native system (e.g. pNFS) is the additional encryption /decryption process which need support from the computation resource of SeDas' client. We compare two systems: (1) A Self-destructing Data System based on Active Storage Framework (SeDas for short), and (2) a conventional system without self-destructing data function (Native for short).

We evaluated the latency of upload and download with two schemes (SeDas and Native) under different file size. Also, we evaluated the overhead of encryption and decryption with two schemes (SeDas and Native) under different file size. Fig. 5(a) shows the latency of the different schemes.We observe that SeDas increases the average latency of the Native system by 59.06% and 25.69% for the upload and download, respectively. The reason for this performance degradation is the encryption and decryption processes introduce the overhead. To illustrate the encryption/decryption latency, Fig. 5(b) plots the overhead of both encryption and decryption processes under different file size in SeDas.



(a) (b)

Fig. 5. Comparisons of latency in the upload and download operations.



(a) (b)

Fig. 6. Comparisons of throughput in the upload and download operations.

Fig. 6 shows the throughput results for the different schemes. The throughput decreases because upload /download processes require much more CPU computation and finishing encryption /decryption processes in the SeDas system, compared with the Native system. From Fig. 6(a), we can see that SeDas reduces the throughput over the Native system by an average of 59.5% and up to 71.67% for the uploading. From Fig. 6(b), we can see that SeDas reduces the throughput over the Native system by an average of 30.5% and up to 50.75% for the downloading.

In summary, the introduced overhead is small: compared with the Native system without self-destructing data mechanism, throughput for uploading and downloading with the proposed SeDas acceptably decreases by less than 72%, while latency for upload/download operations with self-destructing data mechanism increases by less than 60%.

V. CONCLUSION

Data privacy has become increasingly important in Cloud environment. This paper introduced a new approach for protecting data privacy from attackers who retroactively obtain, through legal or other means, a user's stored data and private decryption keys. A novel aspect of our approach is the leveraging of the essential properties of active storage framework based on T10 OSD standard. We demonstrated the feasibility of our approach by presenting SeDas, a proofof-concept prototype based on object-based storage techniques. SeDas causes sensitive information, such as account numbers, passwords and notes to irreversibly self-destruct, without any action on the user's part. Our measurement and experimental security analysis sheds insight into the practicability of our approach. Our plan to release the current SeDas system will help to provide researchers with further valuable experience to inform future object-based storage system designs for Cloud services.

ACKNOWLEDGMENTS

This research is sponsored in part by the National Basic Research 973 Program of China under Grant No. 2011CB302301, and China National Funds for Distinguished Young Scientists under Grant No. 61025008.

REFERENCES

[1] R. Geambasu, T. Kohno, A. Levy, and H. M. Levy, "Vanish: Increasing data privacy with self-destructing data," in *Proc. Of the USENIX Security Symposium*, Montreal, Canada, pp. 299-315, Aug. 2009.

[2] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[3] S. Wolchok, O. S. Hofmann, N. Heninger, E. W. Felten, J. A. Halderman, C. J. Rossbach, B. Waters, and E. Witchel, "Defeating vanish with low-cost sybil attacks against large dhts," in Proc. of the Network and Distributed System Security Symposium, 2010.

[4] L. Zeng, Z. Shi, S. Xu, and D. Feng, "Safevanish: An improved data self-destruction for protecting data privacy," in Proc. of the Second International Conference on Cloud Computing Technology and Science (CloudCom), Indianapolis, IN, pp. 521-528, Dec. 2010.

[5] L. Qin and D. Feng, "Active storage framework for objectbased storage device," in Proc. of the IEEE 20th International Conference on Advanced Information Networking and Applications (AINA), 2006.

[6] Y. Zhang and D. Feng, "An active storage system for high performance computing," in Proc. of the 22nd International Conference on Advanced Information Networking and Applications (AINA), pp. 644-651, 2008.

[7] T. M. John, A. T. Ramani, and J. A. Chandy, "Active storage using object-based devices," in Proc. of IEEE International Conference on Cluster Computing, pp. 472 - 478, 2008.

[8] A. Devulapalli, I. T. Murugandi, D. Xu, and P. Wyckoff. (2009) Design of an intelligent object-based storage device. [Online]. Available: http://www.osc.edu/research/network file/projects/object/papers/istortr.pdf

[9] S. W. Son, S. Lang, P. Carns, R. Ross, R. Thakur, B. Ozisikyilmaz, W.-K. Liao, and A. Choudhary, "Enabling active storage on parallel I/O software stacks," in Proc. of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010.

[10] Y. Xie, K.-K. Muniswamy-Reddy, D. Feng, D. D. E. Long, Y. Kang, Z. Niu, and Z. Tan, "Design and evaluation of oasis: An active storage framework based on t10 osd

standard," in Proc. of the 27th IEEE Symposium on Massive Storage Systems and Technologies (MSST), 2011.

[11] Y. Tang, P. P. C. Lee, J. C. S. Lui, and R. Perlman, "FADE: Secure overlay cloud storage with file assured deletion," in Proc. of the SecureComm, 2010.

[12] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for storage security in cloud computing," in Proc. of IEEE INFOCOM, 2010.

[13] R. Perlman, "File system design with assured delete," in Proc. of the Third IEEE International Security in Storage Workshop (SISW), 2005.

[14] R. Geambasu, J. Falkner, P. Gardner, T. Kohno, A. Krishnamurthy, and H. M. Levy, "Experiences building security applications on dhts," UW-CSE-09-09-01, Tech. Rep., 2009.

[15] Azureus. (2010). Available: http://www.vuze.com/

[16] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "OpenDHT: A public DHT service and its uses," in Proc. of ACM SIGCOMM, 2005.

[17] [Online]. Available: http://www.planet-lab.org/

[18] J. R. Douceur, "The sybil attack," in IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, 2002.

[19] T. Cholez, I. Chrisment, and O. Festor, "Evaluation of sybil attack protection schemes in kad," in Proc. of the 3rd International Conference on Autonomous Infrastructure, Management and Security, Berlin, Heidelberg, 2009, pp.70-82.

[20] B. Poettering. (2006) SSSS: Shamir's secret sharing scheme. [Online]. Available: http://point-at-infinity.org/ssss/

[21] M. Mesnier, G. Ganger, and E. Riedel, "Object-based storage," IEEE Communications Magazine, vol. 41, no. 8, pp.84-90, August 2003.

[22] R. Weber, Information Technology - SCSI Object-Based Storage Device Commands (OSD) -2, Technical Committee T10, INCITS Std., Rev. 5, Jan. 2009.

[23] Y. Lu, D. Du, and T. Ruwart, "QoS provisioning framework for an OSD-based storage system," in Proc. of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST), 2005, pp. 28-35.

[24] Z. Niu, K. Zhou, D. Feng, H. Chai, W. Xiao, and C. Li, "Implementing and evaluating security controls for an objectbased storage system," in Proc. of the 24th IEEE Conference on Mass Storage Systems and Technologies (MSST), 2007.

[25] [Online]. Available: http://www.lustre.org/

[26] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, "Scalable performance of the panasas parallel file system," in Proc. of the 6th USENIX Conference on File and Storage Technologies (FAST), 2008.

[27] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: a scalable, high-performance distributed file system," in Proc. of the 7th symposium on Operating systems design and implementation (OSDI), 2006.

[28] Y. Kang, J. Yang, and E. L. Miller, "Object-based SCM: An efficient interface for storage class memories," in Proc. Of the 27th IEEE Symposium on Massive Storage Systems and Technologies (MSST), 2011.

[29] K. Keeton, D. A. Patterson, and J. Hellerstein, "A case for intelligent disks (IDISKs)," SIGMOD Record, vol. 27, no. 3, September, 1998.

[30] V. Dimakopoulos, A. Kinalis, S. Mastrogiannakis, and E. Pitoura, "The smart autonomous atorage (smas) system," in Proc. of IEEE Pacific Rim Conference on Communications, Computers and signal Processing, pp.303-306, 2001.

[31] E. Riedel, C. Faloutsos, G. Gibson, and D. Nagle, "Active disks for large scale data processing," IEEE Computer, vol. 34, no. 6, pp. 68-74, June 2001.

[32] A. Acharya, M. Uysal, and J. Saltz, "Active disks: Programming model, algorithms and evaluation," in Proc. of the 8th Conference on Architectural Support for Programming Languages and Operating System (ASPLOS), pp. 81-91, October 1998.

[33] G. Chockler and D. Malkhi, "Active disk paxos with infinitely many processes," in Proc. of the 21st Annual Symposium on Principles of Distributed Computing, pp. 78-87, 2002.

[34] R. Wickremesinghe, J. Chase, and J. Vitter, "Distributed computing with load-managed active storage," in Proc. of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC), 2002, pp.13-23.

[35] X. Ma and A. Reddy, "MVSS: an active storage architecture," IEEE Trans. Parallel and Distributed Systems, vol. 14, no. 10, pp. 993-1003, 2003.

[36] M. Wei, L. M. Grupp, F. E. Spada, and S. Swanson, "Reliably erasing data from flash-based solid state drives," in Proc. Of the 9th USENIX Conference on File and Storage Technologies (FAST), San Jose, California, Feb. 2011.

[37] Roadkil's datawipe. [Online]. Available: http://www. roadkil.net/

[38] Secure erase. [Online]. Available: http://cmrr.ucsd.edu/people/Hughes/SecureErase.shtml

[39] Technet sysinternal's sdelete. [Online]. Available:http://technet.microsoft.com

[40] [Online]. Available: http://www.dataerasure.com/recognizedoverwritingstandards.htm

[41] J. L. Sloan. (2011, June) Data remanence and solid state drives. Available: http://coverclock.blogspot.com/2011/06/dataremanence-and-solid-state-disks.html

[42] P. Gutmann, "Secure deletion of data from magnetic and solid-state memory," in Proc. of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography, Berkeley, CA, USA, pp.8, 1996.

[43] P. Gutmann, "Data remanence in semiconductor devices," in Proc. of the 10th conference on USENIX Security Symposium, Berkeley, CA, USA, pp. 4, 2001.

[44] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," Commun. ACM, vol. 52, no. 5, pp. 91-98, 2009.

[45] PNFS. [Online].Available: http://www.pnfs.com/