SeDas: A Self-Destructing Data System Based on Active Storage Framework

Lingfang Zeng[1], Shibin Chen[2], Qingsong Wei[2], and Dan Feng[1]

[1]Wuhan National Laboratory for Optoelectronics, School of Computers, Huazhong University of Science and Technology, 430074 China

[2]Data Storage Institute, A*STAR, 138632 Singapore

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. In addition, 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 evaluations of the *SeDas* prototype, the results demonstrate that *SeDas* is practical to use and meets all the privacy-preserving goals described. Compared to 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*—Active storage, Cloud computing, data privacy, self-destructing data.**

I. Introduction

With development of Cloud computing and popularization of mobile Internet, Cloud services are becoming more and more important for people's life. People are more or less requested to submit or post some personal private information to the Cloud by the Internet. When people do this, they subjectively hope service providers will provide security policy to protect their data from leaking, so others people will not invade their privacy.

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

A pioneering study of Vanish [1] supplies a new idea for sharing and protecting privacy. In the Vanish system, a secret key is divided and stored in a P2P system with distributed hash tables (DHTs). With joining and exiting of the P2P node, the system can maintain secret keys. According to characteristics of P2P, after about eight hours the DHT will refresh every node. With Shamir Secret Sharing Algorithm [2], when one cannot get enough parts of a key, he will not decrypt data encrypted with this key, which means the key is destroyed.

Some special attacks to characteristics of P2P are challenges of Vanish [3], [4], uncontrolled in how long the key can survive is also one of the disadvantages for Vanish. In considering these disadvantages, this paper presents a solution to implement a self-destructing data system, or *SeDas*, which is based on an active storage framework [5]–[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 keys in the object storage system. We use these methods to implement a safety destruct with equal divided key (Shamir Secret Shares [2]).
2) Based on active storage framework, we use an object-based storage interface to store and manage the equally divided key. We implemented a proof-of-concept SeDas prototype.
3) 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. The prototype system imposes reasonably low runtime overhead.
4) SeDas supports security erasing files and random encryption keys stored in a 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.

II. Related Work

A. Data Self-Destruct

The self-destructing data system in the Cloud environment should meet the following requirements: **i**) How to destruct all

Manuscript received November 06, 2012; revised February 17, 2013; accepted February 17, 2013. Date of current version May 30, 2013. Corresponding author: L. Zeng (e-mail: lfzeng@hust.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMAG.2013.2248138

copies of the data simultaneously and make them unreadable in case the data is out of control? A local data destruction approach will not work in the Cloud storage because the number of backups or archives of the data that is stored in the Cloud is 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; **ii**) No explicit delete actions by the user, or any third-party storing that data; **iii**) No need to modify any of the stored or archived copies of that data; **iv**) No use of secure hardware but support to completely erase data in HDD and SSD, respectively.

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 homomorphism authenticator with random mask technique to achieve a privacy-preserving public auditing system for Cloud data storage security and uses the technique of a bilinear aggregate signature to support handling of multiple auditing tasks. Perlman *et al.* [13] present 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 cryptographic techniques with global-scale, P2P, 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 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 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 enough 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, who essentially functions as a trusted third party in this arrangement. It is also susceptible to attacks on roughly 200 PlanetLab [17] nodes on which it runs, most of which are housed low-security research facilities. Vanish is an interesting approach to an important privacy problem, but, in its current form, it is insecure [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 did some improvement on the Shamir Secret Sharing algorithm [20] implemented in the Vanish system. Also, we presented an improved approach against *sniffing attacks* by way of using the public key cryptosystem to prevent from sniffing operations.

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

In addition, for the Vanish system, the survival time of key attainment is determined by DHT system and not controllable for the user. Based on active storage framework, this paper proposes a distributed object-based storage system with self-destructing data function. Our system combines a proactive approach in the object storage techniques and method object, using data processing capabilities of OSD to achieve data self-destruction. User can specify the key survival time of 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 and Active Storage

Object-based storage (OBS) [21] uses an 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 a CPU, network interface, ROM, RAM, and storage device (disk or RAID subsystem) and exports a high-level data object abstraction on the top of device block read/write interface.

With the emergence of object-based interface, storage devices can take advantage of the expressive interface to achieve some cooperation between application servers and 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 about 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.* [25] even implemented the object-based model enables storage class memories (SCM) devices to overcome the disadvantages of the current interfaces and provided new features such as object-level reliability and compression. In recent years, many systems, such as Lustre [26], Panasas [27] and Ceph [28], using object-based technology have been developed and deployed. Since the data can be processed in storage devices, people attempt to add more functions into a storage device (e.g., OSD) and make it more intelligent and refer to it as “Intelligent Storage” or “Active Storage” [5]–[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]–[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 Active Storage Units (ASU). 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 the device level. MVSS separated the deployment of services from file systems and thus allowed services to be migrated to storage devices.

There have been several efforts to integrate active storage technology into the T10 OSD standard. References [5], [7], [8], and [10] all proposed their own implementation of active storage framework for the T10 OSD standard. 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]) in this area by considering data self-destruction.

C. Completely Erase Bits of Encryption Key

In SeDas, erasing files, which include bits (Shamir Secret Shares [2]) of the encryption key, is not enough when we erase/delete a file from their storage media; it is not really gone until the areas of the disk it used are overwritten by new information. With flash-based solid state drives (SSDs), the erased file situation is even more complex due to SSDs having a very different internal architecture [36].

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 they store or the drive in their entirety. 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 which 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.

SSDs work differently than 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 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 as SSDs differ from HDDs in both the technology they use to store data and the algorithms they use to manage and access that data.

Analog sanitization is more complex for SSDs than for hard drives as well. 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, for SSDs, built-in commands are effective, but manufacturers sometimes implement them incorrectly; overwriting the entire visible address space of an SSD twice is usually, but not always, sufficient to sanitize the drive; none of the existing hard drive-oriented techniques for individual file sanitization are effective on SSDs.

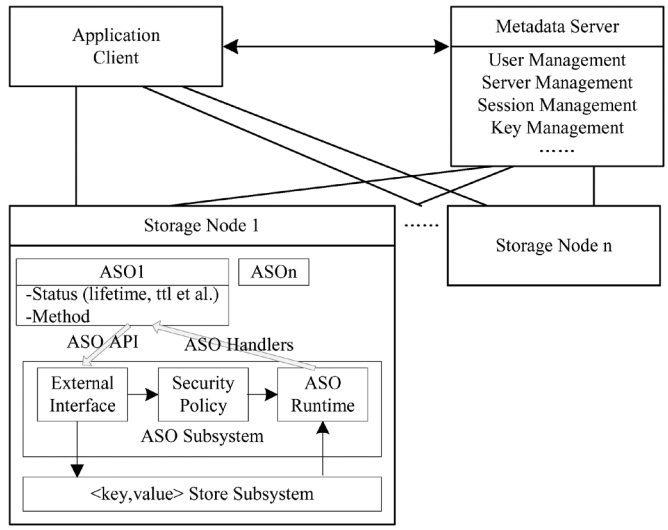

Fig. 1. SeDas system architecture.

To the best of our knowledge, in most of the previous work aimed at some special applications, e.g., database, multimedia, etc., 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 by user controllable key.

III. Design and Implementation of SeDas

A. SeDas Architecture

Fig. 1 shows the architecture of SeDas. There are three parties based on the active storage framework. **i) Metadata server** (MDS): MDS is responsible for user management, server management, session management and file metadata management. **ii) Application node**: The application node is a client to use storage service of the SeDas. **iii) 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 objects 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 and 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 a 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*, *addTime* 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; however, 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 a long time to process a complicated task, so implementing code of a service method in user space can take advantage of 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 the SeDas system, user's applications should implement logic of data process and act as a client node. There are two different logics: uploading and downloading.

i) **Uploading file process** (see Fig. 2): When a user uploads a file to a storage system and stores 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 a 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.

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

In the above code, we assume encrypted data and meta information of the 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 has not 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.

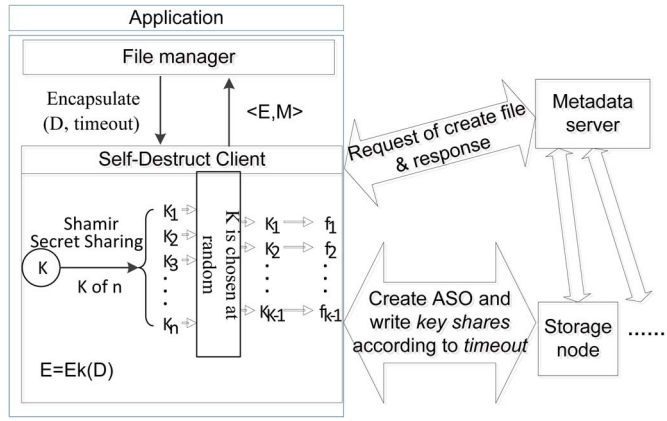

Fig. 2. Uploading file process.

```
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 (pseudo-code).

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 is as follows: **i**) The system prespecifies a directory in a special area to store sensitive files. **ii**) Monitor the file allocation table and acquire and maintain a list of all sensitive documents, the logical block address (LBA). **iii**) LBA list of sensitive documents appear to increase or decrease, the update is sent to the OSD. **iv**) OSD internal synchronization maintains the list of LBA, the LBA data in the list updates. For

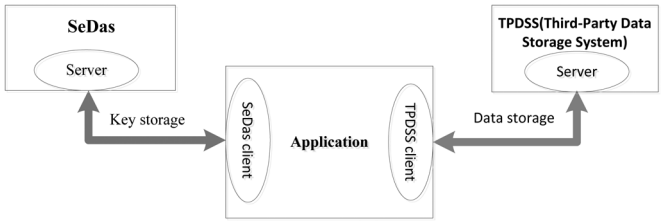

Fig. 4. Structure of user application program realizing storage process.

example, for SSD, the old data page write 0, and then another writes 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 correspond to the page all write. **v**) For ordinary LBA, the system uses the regular update method. **vi**) By calling ordinary data erasure API, we can safely delete sensitive files of the specified directory.

Our strategy only changes a few sensitive documents to the update operation, no effect on the operational performance of the 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 the test for file uploading, downloading and sharing.

A. Experimental Setup and Methodology

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

The way to attain storage service by client interacting with a 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 the decryption is needed after downloading data. In the process of encryption and decryption, the user application program interacts with SeDas. To test the implementation of SeDas described in the previous 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 a 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 Fig. 5.

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.

Device / 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

Fig. 5. Configuration of host and virtual node.

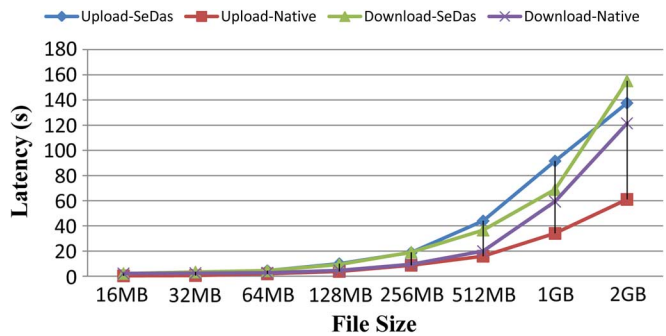

Fig. 6. Comparisons of latency in the upload and download operations.

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 s for a sample text file with 101 bytes. System prompts creating active object are successful afterwards and that means the uploading file gets completed. The time output finally is the time to create active object. SeDas was checked and corresponded with changes on work directory of the storage node. The sample text file also was downloaded or shared successfully before key destruct.

2) Performance Evaluation: As mentioned, the difference of I/O process between SeDas and Native system (e.g. pNFS) is the additional encryption/decryption process which needs support from the computation resource of SeDas' client. We compare two systems: **i**) a self-destructing data system based on active storage framework (SeDas for short), and **ii**) 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 sizes. Also, we evaluated the overhead of encryption and decryption with two schemes under different file sizes. Fig. 6 shows the latency of the

Fig. 7. Comparisons of overhead for encryption and decryption.

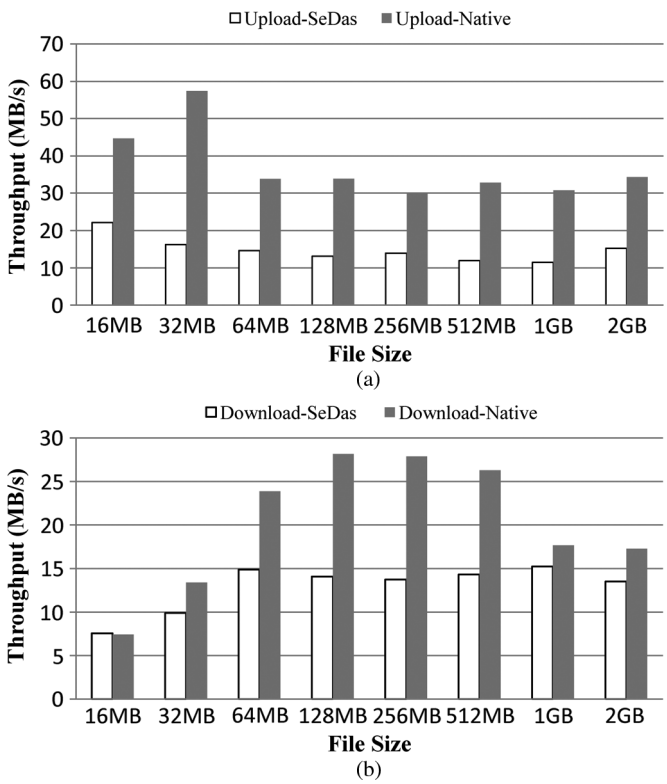

Fig. 8. Comparisons of throughput in the upload and download operations.

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. 7 plots the overhead of both encryption and decryption processes under different file sizes in SeDas.

Fig. 8 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. 8(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. 8(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 the 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 proof-of-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.

Acknowledgment

This work was supported in part by the National 973 Program of China under Grant 2011CB302301, by China National Funds for Distinguished Young Scientists under Grant 61025008, and by A-STAR, Singapore, under Grant 112-172-0010. The authors are grateful to the anonymous reviewers for their valuable comments that helped in improving the paper.

References

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

[2] A. Shamir, "How to share a secret," *Commun. 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 DHEs," in *Proc. Network and Distributed System Security Symp.*, 2010.

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

[5] L. Qin and D. Feng, "Active storage framework for object-based storage device," in *Proc. IEEE 20th Int. Conf. Advanced Information Networking and Applications (AINA)*, 2006.

[6] Y. Zhang and D. Feng, "An active storage system for high performance computing," in *Proc. 22nd Int. Conf. Advanced Information Networking and Applications (AINA)*, 2008, pp. 644–651.

[7] T. M. John, A. T. Ramani, and J. A. Chandy, "Active storage using object-based devices," in *Proc. IEEE Int. Conf. Cluster Computing*, 2008, pp. 472–478.

[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/istor-tr.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. IEEE 26th Symp. 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. 27th IEEE Symp. 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. SecureComm*, 2010.

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

[13] R. Perlman, "File system design with assured delete," in *Proc. Third IEEE Int. Security 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, 2009, Tech. Rep..

[15] Azureus, 2010 [Online]. 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. ACM SIGCOMM*, 2005.

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

[18] J. R. Douceur, "The sybil attack," in *Proc. IPTPS '01: Revised Papers From the First Int. 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. 3rd Int. Conf. Autonomous Infrastructure, Management and Security*, Berlin, Germany, 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 Commun. Mag.*, vol. 41, no. 8, pp. 84–90, Aug. 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. 22nd IEEE/13th NASA Goddard Conf. 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 object-based storage system," in *Proc. 24th IEEE Conf. Mass Storage Systems and Technologies (MSST)*, 2007.

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

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

[27] 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. 6th USENIX Conf. File and Storage Technologies (FAST)*, 2008.

[28] 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. 7th Symp. Operating Systems Design and Implementation (OSDI)*, 2006.

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

[30] V. Dimakopoulos, A. Kinalis, S. Mastrogiannakis, and E. Pitoura, "The smart autonomous atorage (SMAS) system," in *Proc. IEEE Pacific Rim Conf. Communications, Computers and Signal Processing*, 2001, pp. 303–306.

[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, Jun. 2001.

[32] A. Acharya, M. Uysal, and J. Saltz, "Active disks: Programming model, algorithms and evaluation," in *Proc. 8th Conf. Architectural Support for Programming Languages and Operating System (ASPLOS)*, Oct. 1998, pp. 81–91.

[33] G. Chockler and D. Malkhi, "Active disk paxos with infinitely many processes," in *Proc. 21st Annu. Symp. Principles of Distributed Computing*, 2002, pp. 78–87.

[34] R. Wickremesinghe, J. Chase, and J. Vitter, "Distributed computing with load-managed active storage," in *Proc. 11th IEEE Int. Symp. High Performance Distributed Computing (HPDC)*, 2002, pp. 13–23.

[35] X. Ma and A. Reddy, "MVSS: An active storage architecture," *IEEE Trans. Parallel Distributed Syst.*, vol. 14, no. 10, pp. 993–1003, Oct. 2003.

[36] M. Wei, L. M. Grupp, F. E. Spada, and S. Swanson, "Reliably erasing data from flash-based solid state drives," in *Proc. 9th USENIX Conf. File and Storage Technologies (FAST)*, San Jose, CA, USA, 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/recognized_overwriting_standards.htm

[41] J. L. Sloan, June 2011, Data Remanence and Solid State Drives [Online]. Available: http://coverclock.blogspot.com/2011/06/data-remanence-and-solid-state-disks.html