Study & Review of Self Destructing Data System: SeDaS for Secure Cloud Storage

Ms. Dhanashri R. Kulkarni[1], Mr. Hasib M. Shaikh[2]
Department of Computer Engineering,
K. C. College of Engineering & Management Studies & Research,
Kopri, Thane (E),
University of Mumbai, Maharashtra
dhanashri98@gmail.com

Abstract - In cloud storage user may keep their personal passwords, notes, account numbers, and other sensitive and private information that could be accessed or misused by any other unauthorized or illegal user. Most of the times Cloud Service Providers (CSPs') make several copies of user data, catching and archiving of data files without user permission. In the self-destructing data system all the user data and their copies along with decryption key get destructed and unreadable after a user specified time, without any user intervention. All privacy preserving expectations are fulfilled by SeDas with practical use. SeDas throughput for uploading and downloading of data is acceptably decreased while latency for upload/download operations is increased when SeDas is compared with the system without self-destructing data mechanism. Thus SeDas is the most optimum solution to protect important and private data of the user in active storage framework of cloud computing.

Keywords
Self-destructing data, active storage, data privacy, cloud computing.

I. INTRODUCTION

Cloud Computing is a new computing paradigm that is built on virtualization, parallel and distributed computing, utility computing, and service-oriented architecture. Cloud services are becoming more and more important for people's life due to fast development of cloud computing and popularization of mobile internet. One cloud service that is being offered is a revolutionary storage method for user data. From music files to pictures to sensitive documents, the cloud invisibly backs up user files and folders and alleviates the potentially endless and costly search for extra storage space. An alternative to buying an external hard drive or deleting old files to make room for new ones, cloud storage is convenient and cost-effective. It works by storing user's files on a server out in the internet somewhere rather than on local hard drive. This allows user to back up, sync, and access their data across multiple devices as long as user have internet capability. When user use this storage service they must consider the added risk that their information may be accessible to other miscreant, unauthorized user or competitor to whom user do not wish to have access to their data so other people do not invade user privacy.

When cloud storage is used by any user security policy is expected from cloud service provider by the user to protect their data from leaking. On the other hand when the 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 user data privacy.

To provide optimal solution to all these privacy challenges Zeng proposed self destructing data system SeDas [1] which is based on active storage framework of cloud computing. In the SeDas system two new modules are defined, a self-destruct method object that is associated with each secret key part and survival time parameter for each secret key part. In SeDas key distribution algorithm, Shamir's algorithm [2], is used as the core algorithm to implement users distributing keys in the object storage system. An object-based storage interface is also used to store and manage the equally divided key in SeDas. SeDas supports security erasing files and random encryption keys stored in a hard disk drive (HDD) or solid state drive (SSD), respectively. SeDas meet with the requirements of self-destructing data with controllable survival time while users can use this system as a general object storage system. SeDas is practical to use and meets all the privacy-preserving expectations.

A. Data Self Destruct:

All Self-destructing data systems are designed to address personal data security concerns. Their goal is to destroy data after a pre-specified timeout, regardless of where the data is stored or archived and despite technology that may make such deletion challenging. As a result, such systems prevent the exposure of old

data that is past its useful life. Self destruction is implemented by encrypting data with a key and then escrowing the information needed to reconstruct the decryption key with one or more third parties. Assuming that the key reconstruction information disappears from the escrowing third parties at the intended time, encrypted data will become permanently unreadable: (1) even if an attacker obtains a copy of the encrypted data and the users cryptographic keys and passphrases after the timeout, (2) without the user or users agent taking any explicit action to delete it, (3) without needing to modify any stored or archived copies of that data, and (4) without the user relying on secure hardware. Once the key-reconstruction information disappears, data owners can be con_dent that their data will remain inaccessible to powerful attacks, whether from hackers who obtain copies of backup archives and passphrases or through legal means.

B. Object-Based Storage and Active Storage:

Object-based storage is an emerging standard designed to address the problem of data sharing, security, and device intelligence. Object-based storage (OBS) uses an object-based storage device (OSD) as the underlying storage device. A storage object is a logical collection of bytes on a storage device, with well-known methods for access, attributes describing characteristics of the data, and security policies that prevent unauthorized access. Unlike blocks, objects are of variable size and can be used to store entire data structures, such as files, database tables, medical images, or multimedia.

Objects can provide the advantages of both files and blocks. Like blocks, objects are a primitive unit of storage that can be directly accessed on a storage device (i.e., without going through a server); this direct access offers performance advantages similar to blocks. Like files, objects are accessed using an interface that abstracts storage applications from the metadata necessary to store the object, thus making the object easily accessible across different platforms. Providing direct, file-like access to storage devices is therefore the key contribution of object-based storage.

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), security, caching, and backup.

C. Completely Erase Bits of Encryption Key:

As users, corporations, and government agencies store more data in digital media, managing that data and access to it becomes increasingly important. Reliably removing data from persistent storage is an essential aspect of this management process, and several techniques that reliably delete data from hard disks are available as built-in ATA or SCSI commands, software tools, and government standards. In SeDas, erasing files, which include bits 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.

Several techniques that reliably delete data from hard disks are available as built-in ATA or SCSI commands, software tools (such as, DataWipe [3], HDDerase [4] SDelete [5]), and government standards. 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.

The rest of this paper is organized as follows. The review of the literature survey is described in Section II. Section III described methodology and implementation of SeDas. The extensive evaluations results and discussions are presented in Section IV, and Section V concludes this paper.

II. LITERATURE SURVEY

SeDas is self-destructing data system in active storage of cloud computing. To know in detail working of SeDas there should be in detail literature survey of data self-destruct, object-based storage and active storage and completely erase bits of encryption key.

a. Data Self Destruct:

Please us In 2005 Perlman [6] described file system design with assured delete that supported high availability of data, until the data should be expunged, at which time it is impossible to recover the data. This design supported three types of assured delete; expiration time known at file creation, on-demand deletion of individual files, and custom keys for classes of data. The obvious approach, of course, is to encrypt the data on nonvolatile storage, and then destroy keys at the appropriate times.

In 2009 R Geambasu [7] presented Vanish: increasing data privacy with self-destructing data, a system that met with the challenge of privacy preserving through a novel integration of cryptographic techniques with global-scale, P2P, distributed hash tables (DHTs). The work targeted the goal of creating data that self-destructs or vanishes automatically after it is no longer useful. Moreover, it should do so without any explicit action by the users or any party storing or archiving that data, in such a way that all copies of the data vanish

simultaneously from all storage sites, online or offline. The leveraging of the essential properties of modern P2P systems, including churn, complete decentralization, and global distribution under different administrative and political domains was the aspect of this vanish system.

In 2010 L. Zeng [8] proposed a new scheme, called Safe Vanish: improved self data destruction. In Safe Vanish, to prevent hopping attack, which is one kind of the Sybil attacks the length range of the key shares is extended to increase the attack cost substantially, and did some improvement on the Shamir Secret Sharing algorithm are done. Also, an improved approach against sniffing attacks was presented by way of using the public key cryptosystem to prevent from sniffing operations.

In 2010 Tang [9] designed and implemented FADE: secure overlay cloud storage with file assured deletion, a practical, implementable, and readily deployable cloud storage system that focuses on protecting deleted data with policy-based file assured deletion. FADE was built upon standard cryptographic techniques, such that it encrypts outsourced data files to guarantee their privacy and integrity, and most importantly, assuredly deletes files to make them unrecoverable to anyone (including those who manage the cloud storage) upon revocations of file access policies. In particular, the design of FADE was geared toward the objective that it served an overlay system that works seamlessly atop today's cloud storage services, while introducing only minimal performance and monetary cost overhead. Their work provides insights of how to incorporate value-added security features into today's cloud storage services.

In 2010 Wang [10] presented privacy-preserving public auditing for data storage security in cloud computing. In this the public key based homomorphic authenticator was uniquely integrated with random mask technique to achieve a privacy preserving public auditing system for cloud data storage security while keeping all requirements of public auditability for cloud data storage in mind. To support efficient handling of multiple auditing tasks, the technique of bilinear aggregate signature was extended in the main result into a multi-user setting, where TPA could perform multiple auditing tasks simultaneously. Extensive security and performance analysis showed the proposed scheme of Wang was provably secure and highly efficient.

b. Object-Based Storage and Active Storage:

In 2003 M. Mesnier [11] described object-based storage is an emerging standard designed to address the problem of change in the device interface. The author described object-based storage, stressing how it improves data sharing, security, and device intelligence. He also discussed some industry applications of object-based storage and academic research using objects as a foundation for building even more intelligent storage systems. The object interface offers storage that is secure and easy to share across platforms, but also high-performance, thereby eliminating the common trade-off between files and blocks. Furthermore, objects provide the storage device with an awareness of the storage application and enable more intelligence in the device. Object-based storage was designed to exploit the increasing capabilities of storage devices. Characteristics of the future storage device may include self-configuration, self-protection, self-optimization, self-healing, and self-management. Replacing block interfaces with objects is a major step in this evolution.

In 2009 R.Weber [12] has developed working draft on SCSI Object-Based Storage Device Commands (OSD). This SCSI command set is designed to provide efficient operation of input/output logical units that manage the allocation, placement, and accessing of variable- size data-storage containers, called objects. Objects are intended to contain operating system and application constructs. This standard defined the command set extensions to control operation of Object-Based Storage devices.

In 2011 Kang [13] proposed Object-based SCM: An Efficient Interface for Storage Class Memories to fully utilize different characteristics of SCMs, the author proposed the use of an object-based model that provided the hardware and firmware the ability to optimize performance for the underlying implementation, and allowed drop-in replacement for devices based on new types of SCM. The author discussed the design of object-based SCMs and implemented an object-based ash memory prototype. By analyzing different design choices for several subsystems, such as data placement policies and index structures, he showed that their object-based model provides comparable performance to other ash file systems while enabling advanced features such as object-level reliability.

c. Completely Erase Bits of Encryption Key:

In 2011 M. Wei [14] presented Reliably Erasing Data From Flash-Based Solid State Drives. The work focused on empirically evaluation of the effectiveness of hard drive-oriented techniques and of the SSDs built-in sanitization commands by extracting raw data from the SSDs ash chips after applying these techniques and commands. Their results lead to three conclusions: First, built-in commands were effective, but manufacturers sometimes implement them incorrectly. Second, overwriting the entire visible address space of an SSD twice is usually, but not always, sufficient to sanitize the drive. Third, none of the existing hard drive-oriented techniques for individual file sanitization were effective on SSDs. The third conclusion led to develop ash translation layer extensions that exploit the details of ash memory's behavior to efficiently support file sanitization. Overall, they found that reliable SSD sanitization requires built-in, verifiable sanitize operations.

Several techniques that reliably delete data from hard disks are available. Software tools are also available for data delete such as, DataWipe, HDDerase, SDelete, and government standards. All 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 or could be various random patterns. The ATA and SCSI command sets include secure erase commands that should sanitize an entire disk. Physical destruction and degaussing are also effective.

II. METHODOLOGY

Lifetime control over the data is becoming increasingly important as more public and private activities are captured in digital form, whether in the cloud or on personal devices. Self- destructing data systems can help users to preserve some control, by ensuring that data becomes permanently unavailable after a pre-specified timeout. The key properties regarding methodology of SeDas for self-destructing data system based on an active storage framework of cloud computing are described in this section.

a. SeDas Architecture:

The architecture of SeDas is shown in Figure 3.1. 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.

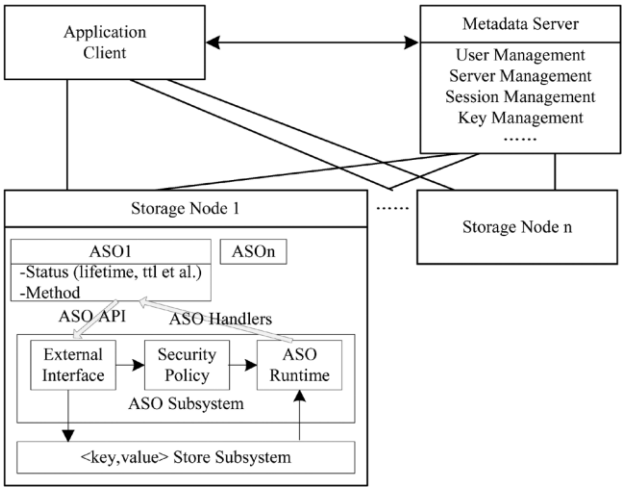

Fig 1: SeDas System Architecture

b. Active Storage Object:

The architecture 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 ttl 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:

The kernel code can be executed efficiently in general; however, a service method is implemented in user space with following considerations in SeDas.

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 applications should implement logic of data process and act as a client node. There are two different logics: uploading and downloading.

Algorithm 1: Uploading File

Require: data (data read from this _le to be uploaded), key (data read from the key), ttl

(time-to-live of the key)

BEGIN

// encrypt the input data with the key

1: bu_er = ENCRYPT(data,key);

2: connect to a data storage server;

3: if failed

4: rEturn fail;

5: create _le in the data storage server and write bu_er into it;

// use ShamirSecretSharing algorithm to get key shares

// k is count of data servers in the SeDas system

6: sharedkeys[1...k] = ShamirSecretSharingSplit(n, k, key);

7: for i from 1 to k

8: connect to DS[i];

9: if successful

10: create object(sharedkeys[i], ttl);

11: else

12: for j from 1 to i

13: delete key shares created before this one;

14: endfor

15: return fail;

16: endif

17: endfor

18: return successful;

END

i) Uploading file process: When a user uploads a file to a storage system and stores his key in this SeDas system, the user should specify the file, the key and ttl as arguments for the uploading procedure. Figure 2 represents the uploading file process. The pseudo-code for uploading file is presented in Algorithm 1. In these codes, 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.

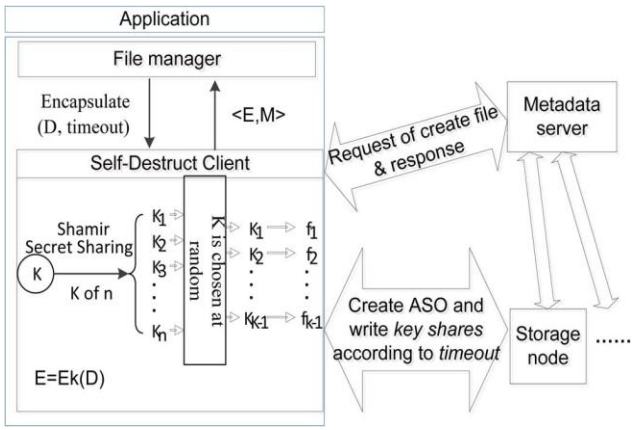

Fig 2: Uploading File Process

In the above code, the encrypted data and metadata 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 client only read encrypted data.

e. Data Security Erasing in Disk:

There must be secure deletion of sensitive data and reduction of 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 updates operation changes, then the OSD performance will be impacted greatly.

In SeDas the implementation method is as follows: i) The system prespecify 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 example, for SSD, the old data page writes 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, the

SeDas system safely delete sensitive files of the specified directory.

Only changes to few sensitive documents are carried out due to the update operation, no effect on the operational performance of the ordinary file in the SeDas. In general, the secure delete function is implied while the OSD read and write performance can be negligible.

f. Security Erasing in Disk:

There are multiple storage services for a user to store data. 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. Figure 3.3 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. 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.

In SeDas the Shamir's Secret Sharing algorithm is used to construct the secret key. The basic concept of this secret sharing is that to divide data D into n pieces in such a way that D is easily reconstructable from any k pieces, but even complete knowledge of k - 1 pieces reveals absolutely no information about D. This technique enables the construction of robust key management schemes for cryptographic systems that can function securely and reliably even when misfortunes destroy half the pieces and security breaches expose all but one of the remaining pieces. Thus this key is generated and used in SeDas system.

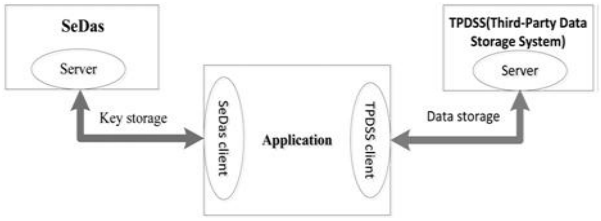

Fig 3: Structure of User Application Program Realizing Storage Process

IV. DISCUSSIONS

To evaluate SeDas, platform was built up on pNFS that supports simple file management, which includes some data process functions such as file uploading, downloading and sharing. Functional testing and performance evaluation of SeDas was done using host and virtual node configurations of SeDas and pNFS. From this evaluation following observations were found out.

1) Functional Testing Observations: After giving the full path of file, key file, and the life time for key parts, the system have encrypted the data and uploaded encrypted data. The life time of key parts was 150s for a sample text file with 101 bytes. System has prompted creating active object was successful afterwards and that means the uploading file got completed. The time output finally was 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. Thus the encryption and decryption was done successfully with key file and life time of key parts. The file was downloaded and shared successfully before key file destruct.

2) Performance Evaluation Observations: There was difference between the I/O process of SeDas and that of Native system (e.g. pNFS). The difference is that the computation resource of SeDas client should support the additional encryption/decryption process. The performance of SeDas was evaluated against Native system (without self-destruct data function). The latency of upload and download with two schemes (SeDas and Native) under different file sizes was evaluated. The overhead of encryption and decryption with two schemes under different file sizes was also calculated. The latency of the SeDas for upload and download operations was more. SeDas have increased 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 was the encryption and decryption processes introduced the overhead. The overhead of SeDas for encryption and decryption under different file system was also increased. The throughput was decreased because upload/download processes required much more CPU computation and finishing encryption/decryption processes in the SeDas system, compared with the Native system. SeDas have reduced the throughput over the Native system by an average of 59.5% and up to 71.67% for the uploading and SeDas have reduced the throughput over the Native system by an average of 30.5% and up to 50.75% for the downloading. All these discussions are compared and summarized in the Table 1.

Table 1. Comparison of SeDas with Native System

Criteria	SeDas	Native System
Overhead	Small increase	Less
Throughput	Decreased	Increased
Latency	Increased	Decreased

V. CONCLUSION

Data privacy and security 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 users stored data and private decryption keys. A novel aspect of SeDas approach is the leveraging of the essential properties of active storage framework based on T10OSD standard. SeDas: self-destructing data system based on active storage framework of cloud computing gives the most optimum solution to protect the important and private data of the user from any kind of attack. SeDas causes sensitive information, such as account numbers, passwords and notes to irreversibly self-destruct, without any action on the users part. This SeDas system will help to provide researchers with further valuable experience to inform future object-based storage system designs for cloud services.

REFERENCES

[1] L. Zeng, S. Chen, and Q. Wei, "SeDas: A self-destructing data system based on active storage framework," ser. 6, vol. 49. IEEE Transactions, June 2013, pp. 2548-2554.

[2] A. Shamir, "How to share a secret," ACM, pp. 612-613, 1979.

[3] R. Datawipe, "Datawipes," http://www.roadkil.net/.

[4] "Secure erase," http://cmrr.ucsd.edu/people/Hughes/SecureErase.shtml.

[5] "Technet sysinternals sdelete," http://technet.microsoft.com.

[6] R. Perlman, "File system design with assured delete," in Third IEEE Int. Security Storage Workshop (SISW), 2005.

[7] R. Geambasu, T. Kohno, and A. A. Levy, "Vanish: Increasing data privacy with self-destructing data," 2009.

[8] 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), 2010, pp. 521-528.

[9] 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.

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

[11] M. Mesnier, G. Ganger, and E. Riedel, "Object-based storage," in IEEE Commun. Mag, ser. 8, vol. 41, August 2003, pp. 84-90.

[12] R. Weber, "Information technology-SCSI object-based storage device commands (OSD)-2," in Technical Committee T10, INCITS Std, January 2009.

[13] 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.

[14] M. Wei, L. M. Grupp, F. E. Spada, and S. Swanson, Reliably erasing data from ash-based solid state drives," in Proc. 9th USENIX Conf. File and Storage Technologies (FAST). San Jose, CA, USA, February 2011.