



SeDas: SELF - DESTRUCTION DATA SYSTEM FOR DISTRIBUTED OBJECT BASED ACTIVE STORAGE FRAMEWORK

N. RamaKalpana¹, R. Santhosh²

Department of Computer Science and Engineering
Faculty of Engineering, Karpagam University, Coimbatore, INDIA

Abstract: *Today's technical and legal landscape presents formidable challenges to personal data privacy. 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. In practical self-destructing data systems that secure sensitive data from disclosure in our highly mobile, social-networked, cloud-computing world. 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 Asymmetric Key techniques with active storage techniques based on T10 OSD standard. 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%.*

Keywords: *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. 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. A goal of creating data that self-destructs or vanishes automatically after it is no longer useful. More-over, 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. 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. 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 the 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. These methods to implement a safety destruct with equal divided key (Shamir Secret Shares [2]).
- Based on active storage framework, we use an object-based storage interface to store and manage the equally divided key.
- 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.
- *SeDas* supports security erasing files and random encryption keys stored in a hard disk drive (HDD) or solid state drive (SSD), respectively.

II. DATA SELF-DESTRUCTION

Self-Destruction data is implemented by encrypting data with a key and that information is needed to reconstruct the decryption key with one or more third parties. 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,(1) even if an attacker obtains a copy of the encrypted data and

the user's cryptographic keys and passphrases after the timeout, (2) without the user or user's agent taking any explicit action to erase it, (3) without needing to modify any saved copies of that data, and (4) without the user relying on secure hardware. The self-destructing data system in the Cloud environment should meet the following requirements:

- i) How to destruct all copies of the data.
- ii) No explicit delete actions by the user, or any third-party storing that data.
- iii) No compelling reason to adjust any of the saved or documented copies of that data.
- iv) No use of secure hardware but support to completely erase data in HDD and SSD, respectively.

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. 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.

III. OBJECT - BASED 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. 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.

IV. 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]. 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.

V. SYSTEM 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.

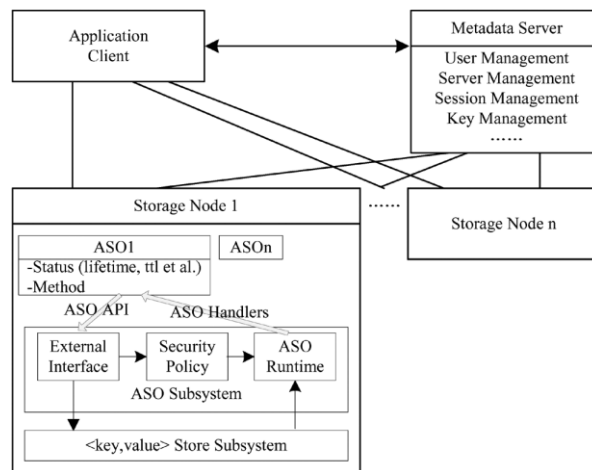


Fig. 1. SeDas system architecture.

VI. 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 *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.

VII. SELF-DESTRUCT METHOD OBJECT

Generally, kernel code can be executed efficiently; 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.

VIII. 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** (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 *tlas* arguments for the uploading procedure. The ENCRYPT procedure uses a common encrypt algorithm or user-defined encrypt algorithm.

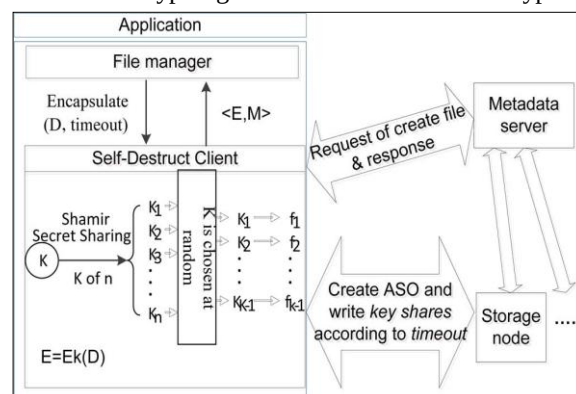


Fig. 2. Uploading file process

After uploading data to storage server, key shares generated by Shamir Secret Sharing 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.

IX. DATA SECURITY ERASING IN DISK

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 pre-specifies 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).

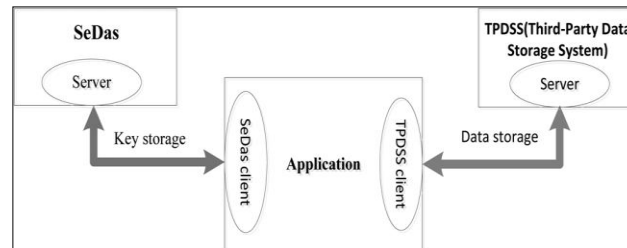


Fig. 3. Structure of user application program realizing storage process

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. 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.

X. ASYMMETRIC CRYPTOGRAPHY

A pair of key namely public and private is used for encryption and decryption respectively. They are different key related to each other mathematically. Each user generates public and private keys. Public key is announced publicly whereas a private key is kept secretly. Each user also maintains a list of public keys of other users. Asymmetric cryptography need not to distribute key since all participants generate public and private keys locally. Plaintext, ciphertext, encryption algorithm, decryption algorithm, public key and private key are the ingredients of asymmetric cryptography. Figure 4 explains the method of asymmetric cryptography. The encryption and decryption of input message is carried as follows:

Encryption: $CT = E(K_{pw} PT)$

Decryption: $PT = D(CT, K_{pr})$

where, K_{pw} and K_{pr} are the public and private key of a user respectively. CT denotes the cipher text; plaintext is expressed as PT. E and D denote encryption and decryption algorithms. A profound example of the asymmetric algorithm is RSA.

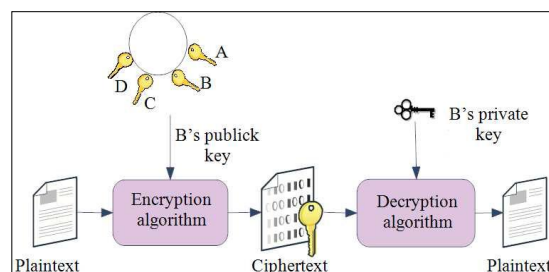


Fig. 4. Asymmetric cryptography

RSA ALGORITHM

Rivest *et al.* (1978) invented RSA algorithm. RSA made use of exponentials. For establishing security in the internet, RSA is used. Its strength is its computational complexity. It is known for its security based on finding the prime factor of very large numbers.

XI. RESULT

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. In this structure (Fig 4), 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 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. The client mainly runs in kernel mode, and it can mount a remote file system to local.

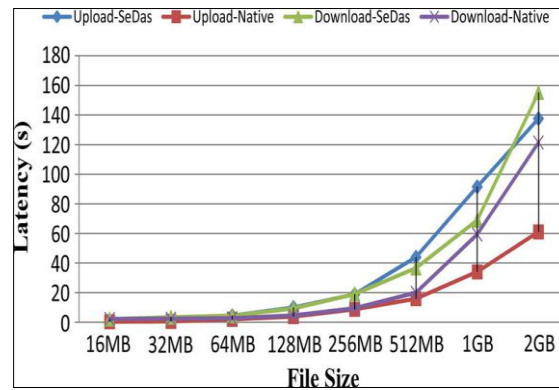


Fig. 5. Comparisons of latency in the upload and download operations

XII.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) ii) A conventional system without self-destructing data function (Native for short).

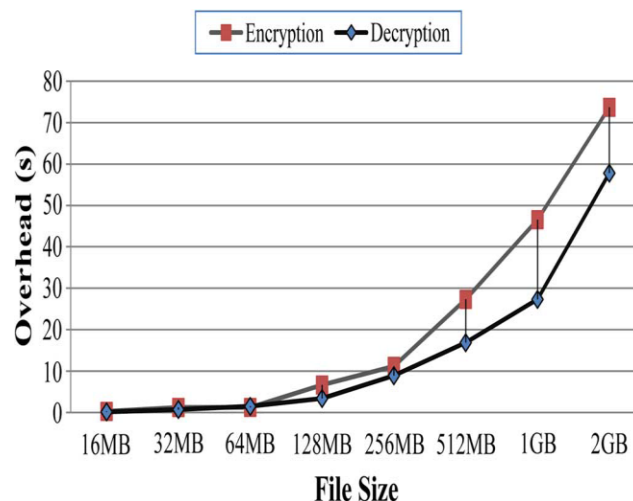


Fig. 6. Comparisons of overhead for encryption and decryption

To illustrate the encryption/decryption latency, Fig. 6 plots the overhead of both encryption and decryption processes under different file sizes in SeDas. Fig. 7 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. 7(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. 7(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.

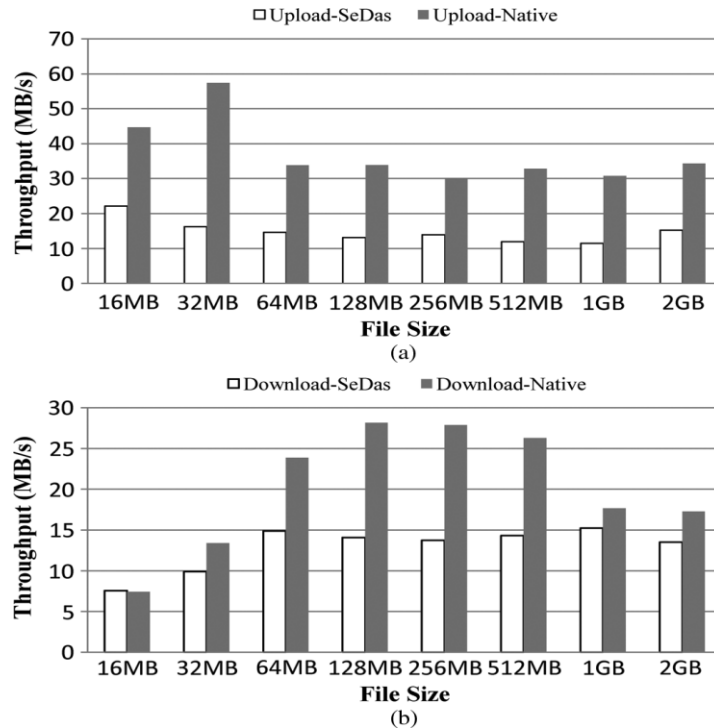


Fig. 7. Comparisons of throughput in the upload and download operations.

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%.

XIII. 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 T10OSD standard. SeDas causes sensitive information, such as account numbers, passwords and notes to irreversibly self-destruct, without any action on the user's part. The fixed data timeout and large replication factor present challenges for a self-destruction data system.

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 panadas 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] Secure Erase [Online]. Available: <http://cmrr.ucsd.edu/people/Hughes/SecureErase.shtml>
- [40] [Online]. Available: http://www.dataerasure.com/recognized_overwriting_standards.htm 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>
- [42] Rivest, R.L., A. Shamir and L. Adleman, 1978. A method for obtaining digital signatures and public key cryptosystems. *Comm. ACM*, 21: 120-126. DOI: 10.1145/359340.359342

About Authors:

First Author Ms. N. RamaKalpana

She is currently a ME Student in Computer Science and Engineering Department at Karpagam University. She received his B.E degree Computer in 2009 from Sasurie College Of Engineering affiliated To Anna University. Her Research area is on Cloud Computing, Data Mining, Image Processing.

Second Author Mr. R.Santhosh

He received his B.Tech degree in information technology from K.S.R College of Technology in 2006, M.E degree in Software Engineering from Sri Ramakrishna Engineering College in 2009, M.B.A in Education Management from Alagappa University in 2011 and pursuing Ph.D in Computer Science and Engineering at Karpagam University. He is currently working as an Assistant Professor in the Department of Computer Science and Engineering at Karpagam University.