ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)
Vol. 2 Issue Special 1 Jan-March 2014
International Journal of Advanced Research in Computer Science & Technology (IJARCST 2014)

SADDs – Self Annihilation and Downloadable Data system in Cloud Storage Service

[I]R.Ramachandran, [II]M.P. Revathi
[I]M.E Dept. of CSE, J J College of Engineering and Technology, Tiruchirappalli, India
[II]Asst Professor (SE G), JJ College of Engineering and Technology, Tiruchirappalli, India

Abstract
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-Annihilating 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. To implement the SADDs security system we are using AES and Random key Generation. Random Key generation is the process of generating keys for cryptography. A key is used to encrypt and decrypt whatever data is being encrypted/decrypted.

Keywords
Active storage, Cloud computing, data privacy, self-Annihilating (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 SADDs, which is based on an active storage framework [5]-[10].

The SADDs 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, SADDs 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 SADDs prototype.
3. Through functionality and security properties evaluation of the SADDs prototype, the results demonstrate that SADDs is practical to use and meets all the privacy-preserving goals. The prototype system imposes reasonably low run-time overhead.
4. SADDs support 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 SADDs 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-annihilating (destructing) data system in the Cloud environment should meet the following requirements: i) How to destruct all 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

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

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 automat-ically self-destruct after a period of time. It integrates crypto-graphic techniques with global-scale, P2P, distributed hash ta-bles (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 Safe Vanish, to prevent hopping attack, which is one kind of Sybil attacks [18], [19], by extending the length range of the key shares to in-crease 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 Safe Vanish, because there is a specific attack against P2P methods (e.g., hopping attacks and Sybil at-tacks [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 pro-poses 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) uses an object-based storage device (OSD) as the underlying storage device. The T10 OSD standard 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), security, caching, and backup. Kang et al. 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 , Panasas and Ceph , 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, SmAS Disk 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 , 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. 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., 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 SADDs, 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 .

Several techniques that reliably delete data from hard disks are available as built-in ATA or SCSI commands, software tools 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. Depending upon the method used, the overwrite data could be zeros (also known as "zero-fill") 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.

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 (over writing 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 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. 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.

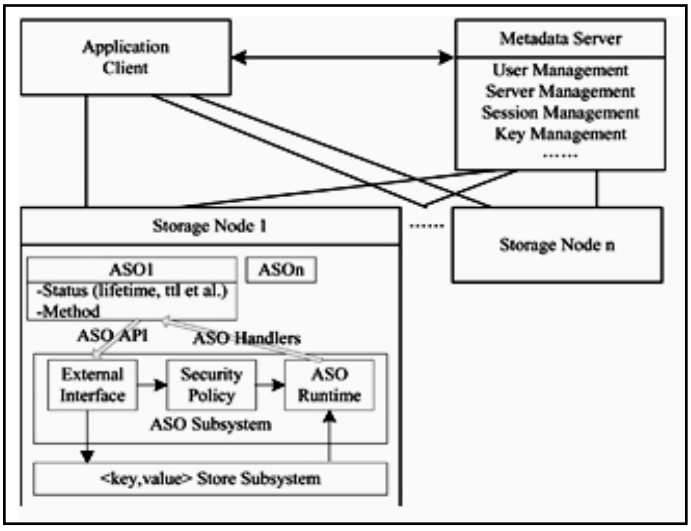

Fig. 1: SADDs System Architecture

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 SADDs, we have implemented a fully functional prototype system. Based on this prototype, we carry out a series of experiments to examine the functions of SADDs. Extensive experiments show that the proposed SADDs 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 SADDs

A. SADDs 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 SADDs. iii) Storage node: Each storage node is an OSD. It contains two core subsystems: key value store subsystem and active storage object (ASO) runtime sub system. 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 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

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

1. Uploading file process (see Fig. 2): When a user uploads a file to a storage system and stores his key in this SADDs system,

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

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 SADDs system.

2. 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.

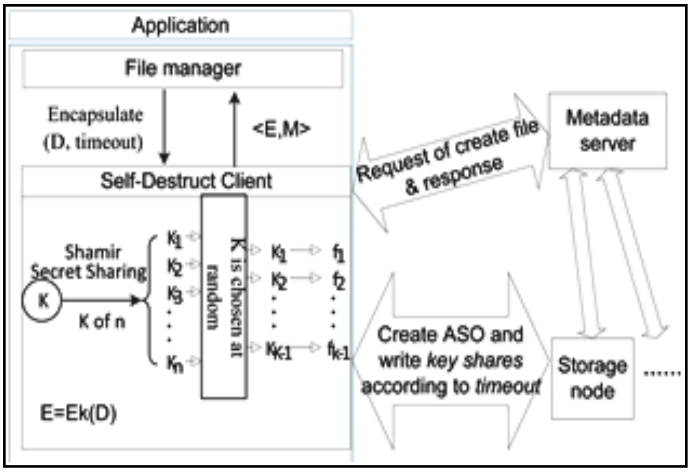

Fig. 2: Uploading File Process

In the pseudo 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 SADDs 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.

```
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 SADD 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
```

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 listof 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 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.

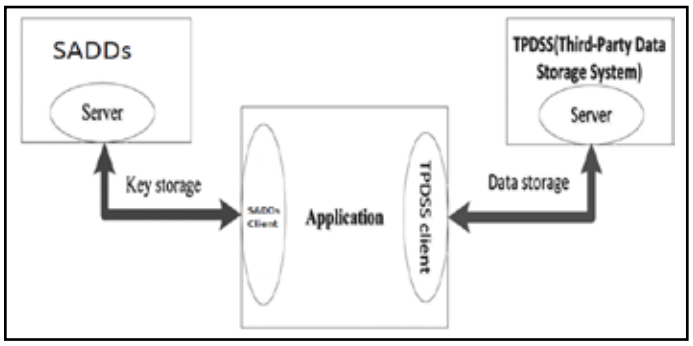

Fig. 4: Structure of user application program realizing storage process

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 lever-aging of the essential properties of active storage framework based on T10 OSD standard. We demonstrated the feasibility of our approach by presenting SADDs, a proof-of-concept prototype based on object-based storage techniques. SADDs 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.

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. Haderman, C. J. Rossbach, B. Waters, and E. Witchel, "Defeating vanishwith 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] 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
[16] 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.
[17] B. Poettering, 2006, SSSS: Shamir's Secret Sharing Scheme [Online].Available: http://point-at-infinity.org/ssss/
[18] Y.Lu,D.Du,andT.Ruwart,"QoSprovisioningframeworkforan OSD based storage system," in Proc. 22nd IEEE/13th NASA Goddard Conf.Mass Storage Systems and Technologies (MSST),2005, pp. 28-35.
[19] 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.
[20] 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.

I am R.Ramachandran. I did my BE. Compter Science in Mohamed sathak Engg college kilakarai and now i am doing my ME. Computer Science in JJ college of Engg and technology. My research area covers security in networking and cloud computing.