www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 3 Issue 5 may, 2014 Page No. 5940-5947

A Systematic Approach to Cloud Security Using SeDas Platform

[1]*Karthik D U*, [2]*Madhu C*, [3]*Sushant M*
[1,2]*PG Student Dept. Of CS&E*, [3]*Assistant Professor Dept. of CS&E*
[1]*Acharya Institute of Technology, Bangalore-107,*
[2,3]*Alvas Institute of Engineering and Technology, Moodbidri-574227*
[1]karthik.mtcs.12@acharya.ac.in, [2]madhuc008@gmail.com, [3]sushantm04@gmail.com

***Abstract--* Cloud computing is one of the cutting-edge technologies used by most of the people. Cloud users are requested to submit the personal private information to the Cloud by the Internet. When users do this, they hope that Cloud service provider (CSPs), will provide the privacy for the data. Self-destructing data mainly aims at providing user's data privacy. All the data and decrypting key will get self-destruct after user specified time without any human intervention. Along with the privacy we can also be possible to achieve the Confidentiality and Integrity. The user data will get encrypted while uploading the file to the cloud using cryptographic algorithms. In order to provide Integrity we use MD5 or SHA algorithm to avoid modification of data. In this paper, we present a system that uses the active storage technique to achieve self-destruction of data based on the T10 OSD standard. A recovery mechanism is also provided to the legitimate users to obtain their data back by requesting to the cloud admin. A new key will be sent to the legitimate user either to the Email or to Mobile using this key he has to login to the SeDas platform to get back their data. So this approach is efficient to use and possible to achieve all the privacy preserving goals described.**

***Index terms-* Active storage, Cloud Computing, Data Privacy, Self Destructing Data.**

I. INTRODUCTION

With the development of Cloud computing and popularization of the mobile Internet, Cloud services are becoming very 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 other's people will not invade their privacy.

As people rely more and more on the Internet and Cloud technology, security of their privacy takes high risks. On the one hand, when data is being processed and stored by the current computer system or network, it should cache, copy and archive it. These copies are essential for systems and the network. However, users don't have knowledge about these copies and cannot control over their data, so these copies may leak their privacy. On the other hand, their privacy also can be leaked via Cloud Service Providers (CSPs'), hackers' intrusion or some legal actions. These problems present challenges to protect people's privacy reconstruction of the original data. Redundancy allows the receiver to detect a limited number of errors without reversing the channel to request the sender for retransmission.

A pioneering study of Vanish [2] supplies a new idea for protecting privacy of the users. In this approach, 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 manage secret keys. According to characteristics of P2P, after about every eight hours the DHT will refresh every node. With Shamir Secret Sharing Algorithm [3], when one cannot get enough parts of a key, it isn't possible to decrypt data encrypted with this key, which indicates that the key is destroyed.

Some special attacks to characteristics of P2P are challenges of Vanish [2], [4], uncontrolled in how long the key can survive is also one of the disadvantages for Vanish. With these disadvantages, this paper presents a solution to implement a self-destructing data system, or SeDas, which is based on an active storage framework [6]–[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.

We focus on the related key distribution algorithm, Shamir's algorithm [3], 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 [3]).Based on the active storage framework; we use an object-based storage to store and manage the equally divided key. We implemented a proof-of-concept SeDas prototype. SeDas supports security erasing files and random encryption keys stored in a hard disk drive (HDD) or solid-state drive (SSD), respectively.

II.RELEATED WORKS

A. Data Self-Destruct

The self-destructing data system [1] 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? Destructing the data locally will not work in the Cloud storage because the number of backups or archives of the data that is stored in the Cloud may not be known, and some nodes preserving the backup data have been offline. The data should become permanently unreadable because of the loss of encryption key, even if an attacker can obtain an original copy of that data; ii) No explicit deletion actions by the user, or any other third-party storing that data; iii) No need to modify any of the stored or archived copies of that data; iv) No need of secure hardware but support to completely erase data in Hard Disk Drive and Solid-State Drive respectively.

ID-based encryption (or identity-based encryption (IBE)) is an important primitive of ID-based cryptography. It is a type of public-key encryption in which the public-key of a user is some unique information about the identity of the user (e.g. a user's email address). This can use the text-value of the name or domain name as a key or the physical IP address it translates to the first implementation of an email-address based PKI was developed by Adi Shamir in 1994, which allowed users to verify digital signatures using only public information such as the user's identifier. If a Private Key Generator (PKG) is compromised, all messages protected over the lifetime of the public-private key pair used by that server are also compromised. This makes the Private Key Generator a high-value target to adversaries.

Tang et al. [12] proposed FADE, which is built upon cryptographic techniques and assuredly deletes files, so it isn't possible to recover them to anyone upon revocation of file access policies. Wang et al. [12] used 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 [ttl] known at file creation, deletion of individual files based on the explicit request, and custom keys for classes of data.

Vanish [5] is a system for creating messages that automatically self-destruct after a certain period of time. It incorporates cryptographic techniques with global-scale, P2P; distributed hash tables (DHTs): DHTs discard data older than a certain period of age. The key is permanently lost, and the encrypted data is permanently unreadable. Vanish works by encrypting each message with a random key and storing shares of the key in a large, public DHT. However, Sybil attacks [4] may compromise the system by continuously crawling the DHT and saving each stored value before it ages out, the total cost is two orders of magnitude less than that mentioned in reference [4] estimated. They can efficiently recover keys for more than 99% of Vanish messages by traversing the DHTs within the certain period of time. Wolchok et a4. [3] concludes that public DHTs like VuzeDHT [05] probably cannot provide strong enough security for Vanish. So, Geambasu et al. [10] proposes two main countermeasures.

Although using both OpenDHT [04] and VuzeDHT might raise the bar for an hacker, at best it can provide the maximum security derived from either system: if both DHTs are not secure, then the hybrid will also be insecure. OpenDHT is controlled by a single maintainer, who functions as a trusted third party in this arrangement. It is also vulnerable to attacks on roughly 200 PlanetLab [09] nodes on which it runs, most of which are housed low-security research facilities. Vanish is an interesting approach to the privacy problem, but, in its current form, it isn't secure.

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 the Sybil attacks [04], by extending the length range of the key shares to in- crease the attack cost, and did some improvement on the Shamir Secret Sharing algorithm [03] implemented in the Vanish system. Furthermore, they presented an enhanced approach against sniffing attacks by way of using the public-key crypto system 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 attacks [5]) .In addition, for the Vanish system, the survival time of key attainment is determined by DHT system and not controllable for the user. This paper proposes a distributed object-based storage system with self-destructing data. The SeDas system combines a proactive approach in the object storage techniques and method object, using data processing capabilities of Object Storage Device to achieve data self-destruction. User can specify the key survival time ttl value of distribution key and use the settings of an expanded interface to export the life cycle of a key, allowing the user to control the life-cycle of private data.

B .Object-based Storage and Active Storage

Object-based storage (OBS) [08] uses an object-based storage device (OSD) [14] as the underlying storage device. The T10 OSD standard [11] is being developed by the Storage Networking Industry Association (SNIA) and the INCITS T10 Technical Commit. 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, active storage devices can take advantage of the expressive interface to achieve some cooperation between application servers and storage devices. It 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 consists of one transaction. Information about data is also stored as objects, which can include the requirements of Quality of Service (QoS) [13], security [4], caching, and backup. Kang et al. [05] 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 [9], Panasas [9] and Ceph [9], using object-based technology have been developed and deployed. Since the data can be processed in storage devices, people can attempt to include more functions into a storage device (e.g., OSD) and make it more intelligent and refer to it as "Intelligent Storage" or "Active Storage" [10]. For example, Intellegent disk [IDISK] [09] and SmAS Disk [10] 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 [11]–[13], but the stream is allowed to pass through only one disk let (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. [11] Proposed a model of load-managed active storage, which is helpful 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 [10], a storage system for active storage devices, provided a single framework to support various services at the device level. It 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 [11] all proposed their own implementation of an 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 [09] and Xie *et al.*'s [10]) in this area by considering data self-destruction.

C. Completely Erase Bits of Encryption

In SeDas, erasing files, which include bits (Shamir Secret Shares [3]) of the encryption key, is not enough when we 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 [6].

Several techniques that reliably delete data from hard disks are available as built-in ATA or SCSI commands, software tools (such as, DataWipe [7], HDDerase [08] SDelete [09]), and government standards (e.g., [04]). These techniques provide effective means of sanitizing HDDs: either individual files they store or the drive. Software methods involve overwriting complete or part of the drive multiple times with patterns specifically designed to obscure any residual data. For instance, different from erasing files, which simply mark file space as available where as the Data Wipe overwrites all data space on a storage device, overwriting the useful data with garbage data. Based upon the method used, the overwrite data could be zeros (also known as "zero-fill") or could be various random patterns [7]. The Advanced Technology Attachment [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 the HDDs, especially when it comes to read and write processes on the drive. The most effective way to delete platter-based HDDs (over- writing space with data) becomes unusable on SSDs because of their design. Data on hard disks can be deleted by overwriting it. This confirms that the data is not recoverable by data recovery tools. This approach will not work 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 [1].

Analog sanitization is more complex for SSDs than for hard drives as well. The analysis in [07] 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. [06] found that, for SSDs, built-in commands are effective, but designers sometimes implement them incorrectly.

Overwriting the entire visible address space of an SSD twice is usually, but it isn't always sufficient to sanitize the drive; none of the existing hard drive-oriented techniques for individual file sanitization are effective on SSDs.

To the best of our knowledge, in most of the previous work aimed at some special applications, example's database, multimedia, etc., there is no system-level self-destructing data in the literature. In 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. The proposed SeDas does not affect the normal use of storage system and can meet requirements of self-destructing data under a survival time by the user controllable key.

III. Design and Implementation of SeDas

A. SeDas Architecture

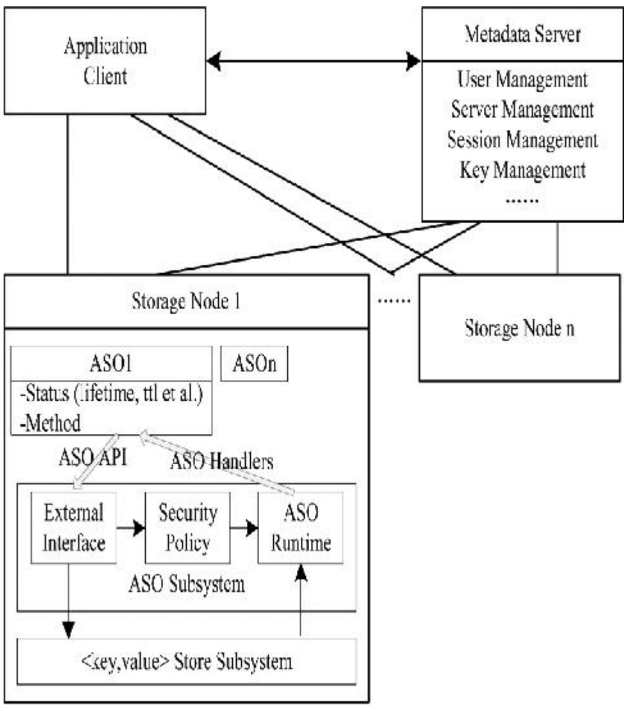


Fig. 1. SeDas system architecture

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 consists of 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 or write object and so on. The object ID is used as a key. The data associated with it, and attribute are stored as values.

The Active Storage Object 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 time to leave 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 Interfaces extended by ActiveStorageObject class are used to manage ttl value.

The create member function needs another argument for ttl. If the argument is one , UserObject:: create will be called to create a user object, else, ActiveStorageOb- ject::create will call UserObject::create first and associate it with the self-destruct method object and a self-destruct policy object with the time to leave 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 are 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; however, a service method should always 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 safer to debug code in user space than in kernel space.

A service method requires a long time to process a complicated task, so implementation of a service method code in user space can take advantage of performance of the system. The system may undergo a 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 requires three arguments. The lun argument indicates the device; the pid argument specifies the partition, and the obj_id argument specifies the object to be destructed.

D. Data Process

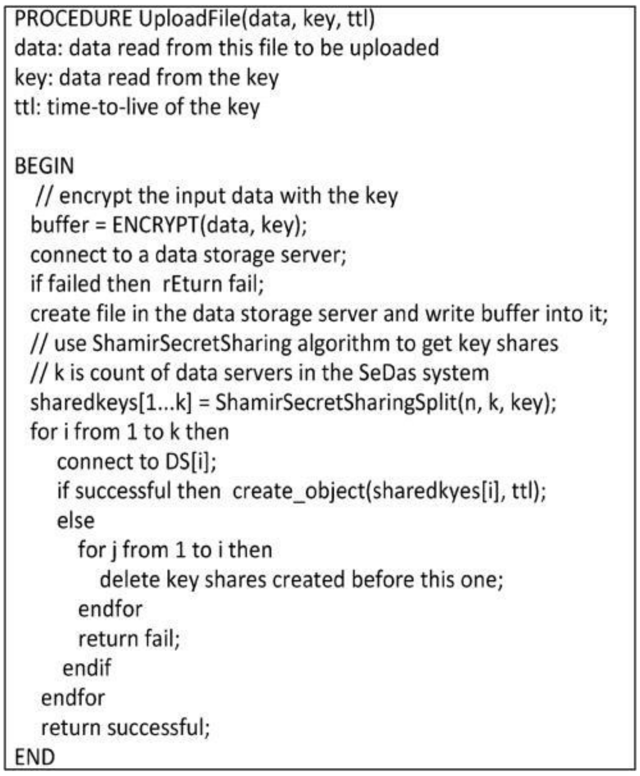
To use the SeDas system, user's applications should implement logic of data process and act as a client node. There are two different logics: uploading and downloading.

I. Uploading the File (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 indicates its pseudo-code. In these codes, we assume data and key has been read from the file. The procedure ENCRYPT uses a common encrypt algorithm or user-defined encrypt algorithm. After uploading user's data to storage server, key shares generated by SSS algorithm will be used to create an 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 entire logic is implemented in code of user's application. In the above code, we assume encrypted data and Meta's information of the key has been read from the downloaded file. Prior 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 users can get enough key shares to reconstruct the key successfully. If the associated ASO of that key.



```
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.2 uploading the file (pseudo-code)

E. Data Security Erasing Bits

We must secure delete sensitive data and reduce the negative impact of OSD performance due to deleting operation. The pro- portion of required secure deletion of all the files is not great, so if this part of the file updates operation changes, then the performance of the OSD will be impacted. The implementation method is as follows: i) the system respecify a directory in a special area to store sensitive files ii) Monitor the FAT and acquire and maintain a list of all sensitive documents, the logical block address (LBA). iii) Logical Block Address consists the list of sensitive documents appears to increase or decrease; the update is sent to the OSD. iv) The internal synchronization maintains the list of LBA, the data in the list updates. For SSD, the old data page writes 0, and then another writes the new data page. When the LBA list is shorter than that of the file, size is shrinking. At this time, the old data needs to be corresponding to the page all write. v) For ordinary Logical Block Address, the system uses the regular update method. vi) By calling data erasure API, we can securely delete sensitive files of the specified directory.

Our strategy only changes a few sensitive documents to the update operation; it will not effect on the operational performance of the file. In general, the secure delete function is implied while the OSD read and write performance can be negligible.

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 or write object and so on. The object ID is used as a key. The corresponding data and attribute are stored as values. A service method needs a long time to process a complex task, so implementing code of a service method in user space can take advantage of performance of the system.

IV. OVERALL ARCHITECTURE AND WORKING FLOW

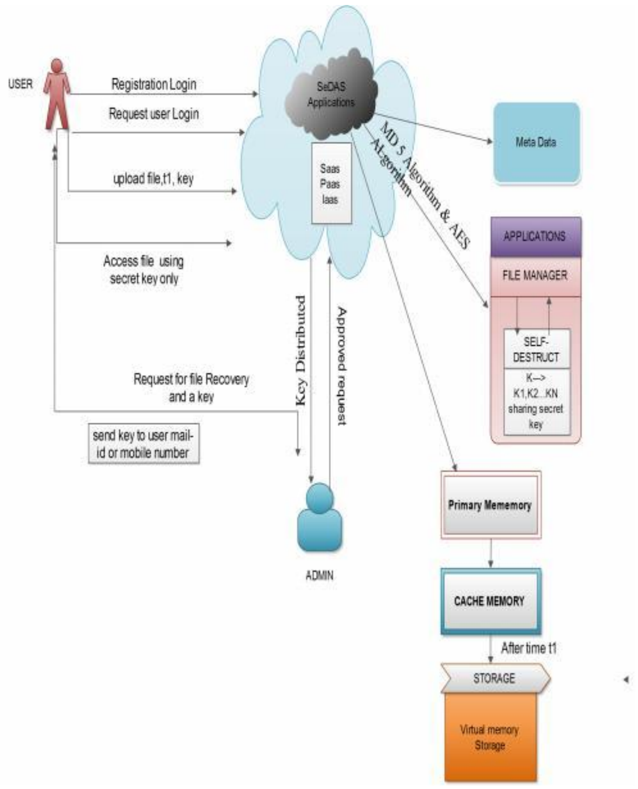


Fig 3. Overall Architecture

The fig 3 and fig 4 shows the overall architecture and working flow. The SeDas platform is available within the cloud. The cloud users first register to the SeDas Platform. Once users get registered, now they can be able to login, and they can perform the file operation, i.e. file upload or download in the cloud. Now user data will get to encrypt and decrypt each time when he made the file operation. After the TTL value, the data will get automatically destroy and that will be moved to the cache memory. The data present in the cache will be moved to the virtual memory, a special directory which is under the control of Admin. If the user wants that particular data then he has to send the request to admin.

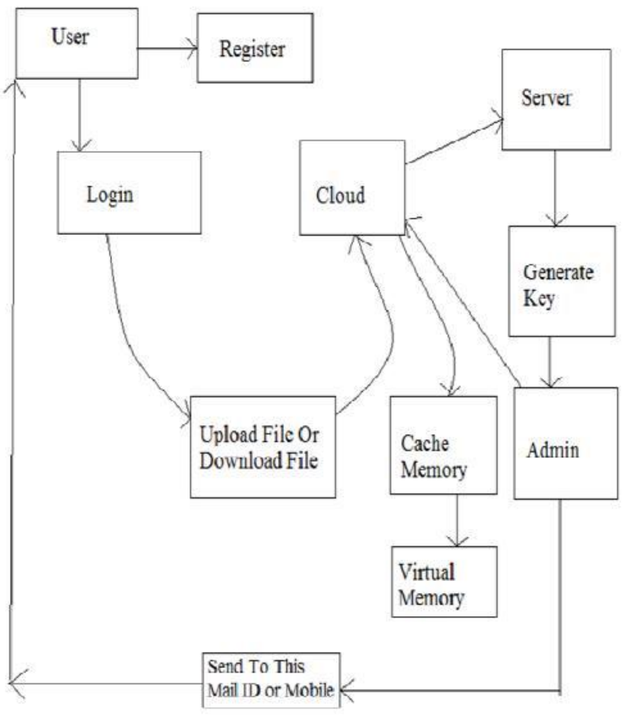


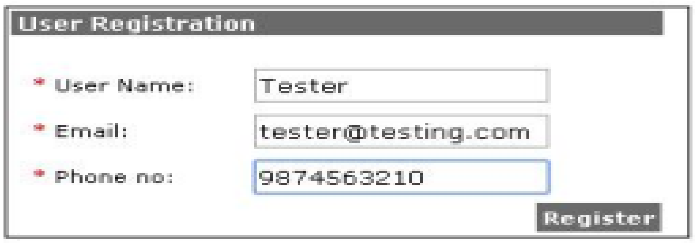
Fig 4. Working Flow

After the proper authentication of the user's credentials, the admin will send that key to the user. The AES algorithm will provide key and this key will be sent to the admin. User will login using that key to get back his original data. Here we are using the MD5 or SHA algorithm while providing the data to the user in order to provide the integrity of the file. The key will be sent to both user's mail-id and mobile number. Once again, the user has to login using the recovery key to get back his original data.

Meta data module which consists of some basic information such as the type of algorithm used for the encryption, algorithm used for the key generation and key sharing, type of the database used et.al.

V. RESULTS

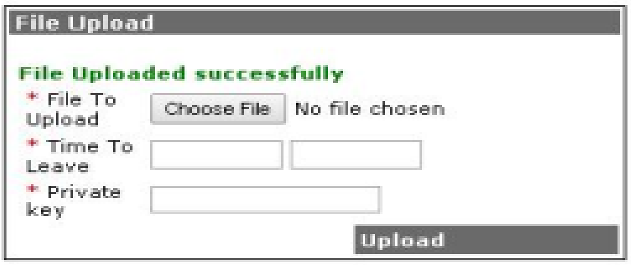
1. User registers to SeDas platform.


User Registration

* User Name:	Tester
* Email:	tester@testing.com
* Phone no:	9874563210

User login into SeDas platform using username and password which is sent to the Email id.

2. User uploads the file.

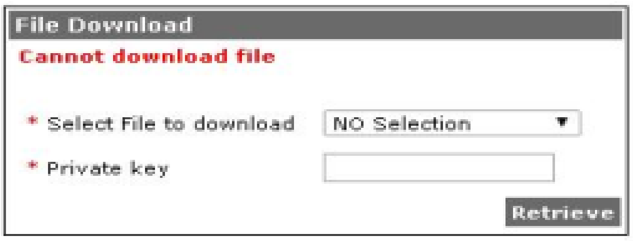

File Upload

File Uploaded successfully

* File To Upload	Choose File No file chosen
* Time To Leave	
* Private key	

Once the user credentials are verified, he can able to upload the file to the SeDas platform. The time to live and the private key for encryption fields should be provided by the user while uploading the file.

3. Once the TTL expires file cannot be download.

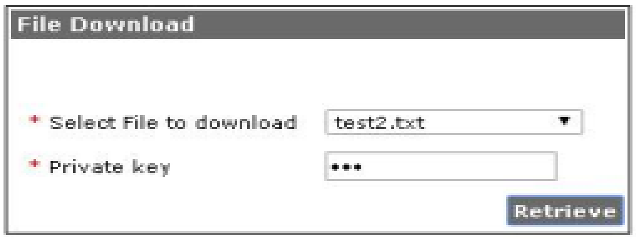

File Download

Cannot download file

* Select File to download	NO Selection
* Private key	

After the timeout occurs the users uploaded files will get deleted.

4. File downloads after admin approval.


File Download

* Select File to download	test2.txt
* Private key	•••

After TTL expires, user has to get the permission from the admin to recover back his file.

VI. CONCLUSION

Data privacy has become increasingly important in the Cloud environment. This paper introduced an object-based methodology for protecting data privacy from attackers who obtain data, through legal or other means, a user's stored data and private decryption keys. The main aim of our approach is to utilize the essential properties of an active storage framework based on T10 OSD standard. SeDas causes sensitive information, such as account numbers, passwords and notes to irreversibly self-destruct, without any action on the user's part. Along with the privacy, this paper will also provide the integrity by using the MD5 or SHA algorithm and the confidentiality by encrypting the user data before entering into the cloud. We can achieve greater security in this approach by allowing the users to specify the survival time of the key [TTL] which allows the user to control the life-cycle of the private data.

REFERENCES

[1] Lingfang Zeng, shibin Chen, Qingsong Wei and Dan Feng, "SeDAS: A Self-Destructing Data System Based On Active Storage Framework," IEEE Transaction on Magnetics, vol 49, No.6, June 2013

[2] 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.

[3] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[4] S. Wolchok, O. S. Hofmann, N. Heninger, E. W. Felten, J. A. Hal- derman, 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] 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/ob- ject/papers/istor-tr.pdf

[10] 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.

[11] 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. Mas- sive Storage Systems and Technologies (MSST)*, 2011.

[12] Y. Tang, P. P. C. Lee, J. C. S. Lui, and R. Perlman, "FADE: Se- cure overlay cloud storage with file assured deletion," in *Proc. SecureComm*, 2010.

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

[14]. M. Mesnier, G. Ganger, and E. Riedel, "Object-based storage," *IEEE Commun. Mag*[19] T. Cholez, I. Chrisment, and O. Festor, "Evaluation of sybil attack pro- tection schemes in kad," in *Proc. 3rd Int. Conf. Autonomous Infrastruc- ture, Management and Security*, Berlin, Germany, 2009, pp. 70–82., vol. 41, no. 8, pp. 84–90, Aug.