International Journal of Scientific Engineering and Research (IJSER)
www.ijser.in
ISSN (Online): 2347-3878
Volume 2 Issue 3 March 2014

Sedas: A Self Destructive Active Storage Framework for Data Privacy

R. C. Dharmik[1], Hemlata Dakhore[2], Vaishali Jadhao[3]

[1]Department of IT YCCE, Nagpur, Maharashtra, India

[2]Department of CSE, G.H Raisoni Nagpur, Maharashtra, India

Abstract: ***Personal data that we store 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 and copied by Cloud Service Providers, often without users' authorization and control. Self-destructing data mainly aims at protecting the user data's privacy. All the data and their copies become destructed or unreadable after a user-specified time, without any information to user. In addition, the decryption key is destructed after the user-specified time. In this paper, we present a system that meets this challenge through a novel integration of cryptographic techniques with active storage techniques. We implemented a proof-of-concept self destructive prototype. Through functionality and security properties evaluations of this prototype, the results demonstrate that the system is practical to use and meets all the privacy-preserving goals described. Compared to the system without self-destructing data mechanism, throughput for uploading and downloading with the proposed system acceptably decreases, while latency for upload/download operations with self-destructing data mechanism increases.***

Keywords: Active storage, Cloud computing, data privacy, self-destructing data

1. Introduction

As Cloud computing and mobile Internet is getting popularized, Cloud services are becoming more and more important in people's life. People are requested to submit or post some personal private information to the Cloud by the Internet. When people post their data, 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 is on 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. Because these copies are essential for systems and the network. As 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 negligence, hackers' intrusion or some legal actions. These problems present formidable challenges to protect people's 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, often without users' authorization and control. Self-destructing data mainly aims at protecting the user data's privacy. All the data and their copies become destructed or unreadable after a user-specified time, without any user intervention. In addition, the decryption key is destructed after the user-specified time.

2. Objectives

Objectives of Proposed System to implement a self destructing data system are as follows:

The self destructive system defines two 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, System 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 objectives are summarized as follows.

1) We focus on the related key distribution algorithm, Shamir's algorithm, 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.
2) Based on active storage framework, we use an object-based storage interface to store and manage the equally divided key.
3) Through functionality and security properties evaluation of this prototype, the results demonstrate that System is practical to use and meets all the privacy-preserving goals. The prototype system imposes reasonably low runtime overhead.
4) System supports security erasing files and random encryption keys stored in a hard disk drive or solid state drive, respectively.
5) Using load balancing and round robin algorithm for managing the load on the nodes,

3. Related Work

3.1 Existing System

A pioneering study of Vanish supplies a new idea for sharing and protecting privacy. In the Vanish system, a secret key is divided and stored in a point to point system with distributed hash tables. With joining and exiting of the point to point node, the system can maintain secret keys. According to characteristics of point to point, the distributed hash tables will refresh every node after every eight hours. With Shamir Secret Sharing Algorithm, when we will not get enough parts of a key, he will not decrypt data encrypted with this key, which means the key is destroyed and the data cannot be recovered. Some special attacks to characteristics of point to point are challenges of Vanish, uncontrolled in how long the key can survive.

Vanish is a system used for creating messages that automatically self-destruct after a period of time. It integrates cryptographic techniques with global-scale, point to point distributed hash tables. Distributed hash tables have the property to discard data older than a certain age. In this the key is permanently lost, and the encrypted data is permanently unreadable after data expiration time. In Vanish system each message is encrypted with a random key and storing share of the key in a large, public distributed hash tables. However, Sybil attacks may compromise the system by continuously crawling the distributed hash tables and saving each stored value before it ages out and the total cost is two orders of magnitude less than that mentioned in estimated. They can efficiently recover keys for more than 99% of Vanish messages.

3.2 Proposed System

The self destructive system defines two new modules, a self-destruct method object that is associated with each secret key part and each secret key part has its own survival time parameter. In this case, self destructive system can meet the requirements of self-destructing data with controllable survival time while users can use this system as a general object storage system. We are using load balancing and round robin algorithm for managing the data on the nodes.

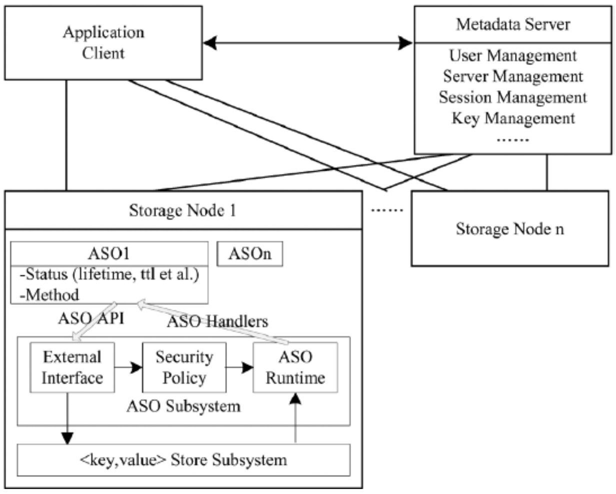

Figure 1: Self destructive system architecture

A) Self destructive Architecture

Figure 1 shows the architecture of self destructive. There are three parties based on the active storage framework.

i. **Metadata server**: It 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 self destructive.
iii. **Storage node**: Each storage node is an OSD. It contains two core subsystems: key value store subsystem and active storage object runtime subsystem. The key value store subsystem which is based on the object storage component and 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 of the node 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 value property. The time-to-live value is used to trigger the self-destruct operation. The time-to-live *value* of a user object has the property infinite so the user object will not be deleted until a user deletes it manually. On the other hand the time-to-live *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.

C) Self-Destruct Method Object

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 self destructive 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:** When a user uploads a file to a storage system and stores his key in this System, he has to specify the file, the key and time-to-live 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, the key shares that are generated by *Shamir Secret Sharing* algorithm are used to create active storage object in storage node in the Self destructive 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.

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 these parts 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.
iii) Logical block address list of sensitive documents appear to increase or decrease, the update is sent to the OSD.
iv) OSD internal synchronization maintains the list of logical block address, the logical block address data in the list updates.

4. Proposed Plan of Work

1. **Study of the Existing System as well as Proposed System**
 In this module we will do study of the existing system and also of the proposed system and whatever disadvantage that are in the existing system we have to remove it and have to see that it does not occur in the proposed system.
2. **Development of Active Storage Framework**
An active storage object that is derived from a user object and has the time-to-live value property. The time-to-live value is used to trigger the self-destruct operation. The time-to-live value of a user object is infinite i.e. user object will not be deleted until a user deletes it manually. The time-to-live 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.
3. **Development of login tracking of the user**
To use the Self destructive 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: When** a user uploads a file to a storage system and stores his key in this System, he should specify the file, the key and time-to-live *as* arguments for the uploading procedure. 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 *Shamir Secret Sharing* algorithm will be used to create active storage object in storage node in the self destructive 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.
4. **Development of deletion module in case user logs out**
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.
5. **Checking the response of the system**
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 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).
   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.
6. **Testing and optimization of our system**
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. Self destructive 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.

5. Research Methodology

We focus on the related key distribution algorithm, Shamir's algorithm, 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) 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 Self destructive prototype. Through functionality and security properties evaluation of the Self destructive prototype, the results demonstrate that self destructive is practical to use and meets all the privacy-preserving goals. The prototype system imposes reasonably low runtime overhead. 4) Self destructive supports security erasing files and random encryption keys stored in a hard disk drive or solid state drive, respectively.

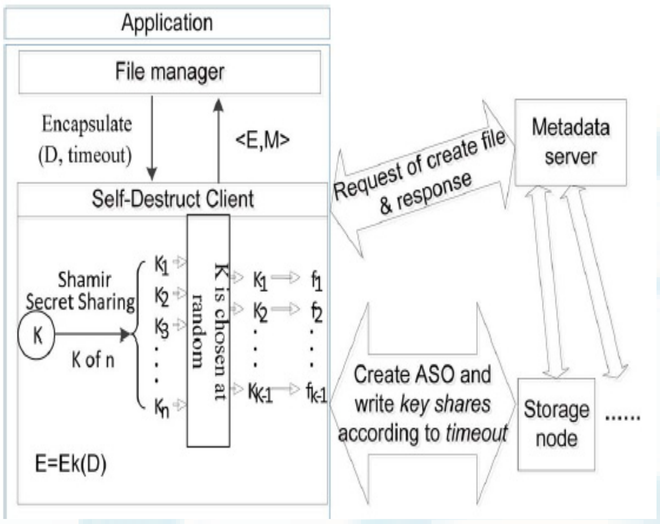

Figure 2: Key distribution using Shamir secret sharing algorithm

6. 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.

References

[1] IEEE paper on "SeDas: A Self-Destructing Data System Based on Active Storage Framework" by: Lingfang Zeng, Shibin Chen, Qingsong Wei, and Dan Feng IEEE TRANSACTIONS 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] 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.

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