International Journal of Information & Computation Technology.
ISSN 0974-2239 Volume 4, Number 11 (2014), pp. 1035-1042
© International Research Publications House
http://www. irphouse.com

SADS – Self Annihilating Data Storage system in Cloud Storage Service

Mohan Sadasivam[1], Rajeeve Dharmaraj[2]

[1,2]M.Tech Information and Communication Technology, JJCET Trichy
Anna University – Chennai, 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 SADS 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

1. 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 SADS, which is based on an active storage framework [5]-[10].

The SADS 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, SADS 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 SADS prototype.
3) Through functionality and security properties evaluation of the SADS 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) SADS supports 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. And we conclude this paper in Section IV.

2. Related Work

2.1 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 ap-proach 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 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.

2.2 . Object-Based Storage and Active Storage

Object-based storage (OBS) [14] uses an object-based storage device (OSD) [15] as the underlying storage device. The T10 OSD standard [15] 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) [16], security [17], caching, and backup. Kang et al. [18] 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 [19], Panasas [20] and Ceph [21], 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, IDISK [22] and SmAS Disk [23] 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 [24][25], but the stream is allowed to pass through only one disklet (user-specific code).

3. Design and Implementation of Self Annihilating Data Storage

3.1 SADS Architecture

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.

3.2 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.

3.3 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.

3.4 Data Process Evaluation and Discussion

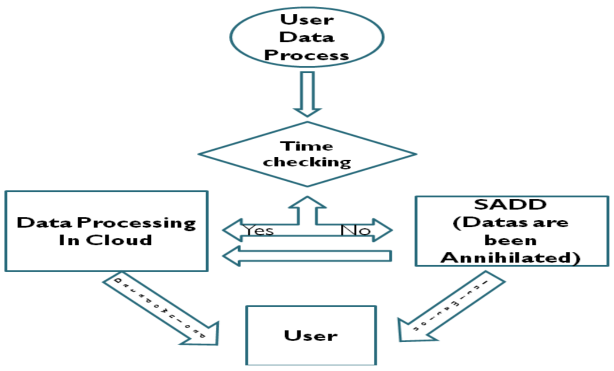

Figure 1: Function flow of SADS

Whenever the data is processed in the cloud. The user predefined time is checked whether the time is exceeded or not. If the time is not exceeded, the data is processed as given in the diagram. Or else the SADS function is called so that the Self Annihilation function is employed. All the data that are been stored in the cloud will be destructed.

4. 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. SADS causes sensitive information, such as account numbers, passwords and notes to irreversibly self-destruct, without any action on the user's part.

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] M. Mesnier, G. Ganger, and E. Riedel, "Object-based storage," IEEE Commun. Mag., vol. 41, no. 8, pp. 84-90, Aug. 2003.

[15] R. Weber, "Information Technology—SCSI object-based storage device commands (OSD)-2," Technical Committee T10, INCITS Std.,Rev. 5 Jan. 2009.

[16] Y.Lu,D.Du,andT.Ruwart,"QoSprovisioningframeworkforanOSD based storage system," in Proc. 22nd IEEE/13th NASA Goddard Conf.Mass Storage Systems and Technologies (MSST),2005, pp. 28-35.

[17] 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.

[18] 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.

[19] [Online]. Available: http://www.lustre.org/

[20] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, “Scalable performance of the panasas parallel file system,” in Proc. 6th USENIX Conf. File and Storage Technologies (FAST), 2008.

[21] 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.

[22] K. Keeton, D. A. Patterson, and J. Hellerstein, “A case for intelligent disks (IDISKs),” SIGMOD Rec., vol. 27, no. 3, Sep. [23] 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.

[24] 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.

[25] 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 (AS-PLOS), Oct. 1998, pp. 81-91.