SDOS: Using Trusted Platform Modules for Secure Cryptographic Deletion in the Swift Object Store

Tim Waizenegger
University of Stuttgart
Institute for Parallel and Distributed Systems
waizentm@ipvs.uni-stuttgart.de

Frank Wagner
University of Stuttgart
Institute for Parallel and Distributed Systems
wagnerfk@ipvs.uni-stuttgart.de

Cataldo Mega
University of Stuttgart
Institute for Parallel and Distributed Systems
megaco@ipvs.uni-stuttgart.de

ABSTRACT

The secure deletion of data is becoming increasingly important to individuals, corporations as well as governments. Recent advances in worldwide laws and regulations now require secure deletion for sensitive data in certain industries. Data leaks in the public and private sector are commonplace today, and they often reveal data which was supposed to be deleted. Secure deletion describes any mechanism that renders stored data unrecoverable, even through forensic means. In the past this was achieved by destroying storage media or overwriting storage sectors. Both of these mechanisms are not well suited to today's multi-tenant cloud storage solutions.

Cryptographic deletion is a suitable candidate for these services, but a research gap still exists in applying cryptographic deletion to large cloud storage services. Cloud providers today rarely offer storage solutions with secure deletion for these reasons. In this Demo, we present a working prototype for a cloud storage service that offers cryptographic deletion with the following two main contributions:

A key-management mechanism that enables cryptographic deletion an on large volume of data, and integration with Trusted Platform Modules (TPM) for securing master keys.

Keywords

secure data deletion, cryptographic deletion, data erasure, records management, retention management, key management, data shredding, trusted platform module, TPM

1. BACKGROUND

Cloud based storage solutions are popular services today especially among consumers. They are used for synchronizing data across devices, for backup and archiving purposes, and for enabling access at any time from anywhere. But the adoption of such storage services still faces many challenges in the government and enterprise sector. The customers, as well as the providers, have a desire to move storage systems, or parts of these systems, to cloud environments in order to reduce cost and improve the service. But security issues often prevent customers from adopting cloud storage services.

The providers often address these issues by offering some type of data encryption. They differ in three aspects: i) where the data encryption happens, ii) who has authority over the encryption keys, and iii) how keys are managed.

In most offerings, the provider has authority over master keys and encryption happens on the provider side [1]. This allows the provider to read the customer's data and enables them to offer more advanced services and up-sell customers in the future. If client side encryption is used and customers have authority over master keys, no provider access is possible and less trust in the provider is required. Client side encryption with customer side key authority enables the use of cloud storage services for especially sensitive data.

We propose a cloud storage systems that employs client-side encryption of content in order to address confidentiality concerns of customers. We further propose a key management that enables cryptographic deletion in order to assure customers' legal and regulatory compliance.

In this demo, we present a cloud storage system with the two main contributions:

1. Transparent data encryption with support for cryptographic deletion.
2. Trusted Platform Module integration that provides secure deletion and confidentiality for master keys.

2. CRYPTOGRAPHIC DELETION

An often overlooked security aspect of cloud storage systems is the secure deletion of data. Secure deletion describes any mechanism that renders deleted data unrecoverable, even through forensic means. Recent advances in worldwide regulation make secure deletion a requirement in many industries like banking and law enforcement [2, 4, 5]. Even industries without explicit regulation have an interest in securely removing deleted data in order to prevent future leaks and exposure [6]. In the past, secure deletion was achieved by destroying storage media or overwriting storage sectors. Both of these mechanisms are not well suited to today's multi-tenant cloud storage solutions. Identifying the physical disks that need to be destroyed, or the blocks that need to be overwritten, becomes difficult to impossible [7]. In this work, we assume an untrusted cloud storage

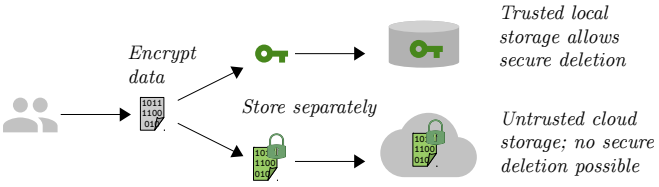

Figure 1: Cryptographic deletion: Delete data indirectly by securely deleting their encryption key.

system in which all deleted data can be restored later by forensic means. Cryptographic deletion is a suitable candidate for these services, but a research gap still exists in applying cryptographic deletion to large cloud storage services [8]. As shown in Figure 1, cryptographic deletion works by encrypting the data prior to storing it in an untrusted location. The encryption key must be kept in a secure location and can later be deleted in order to indirectly delete the data. This requires that data encryption happens in the customers trusted environment and that the customer has authority over the encryption key. Cryptographic deletion is therefore a function of the key management mechanism in client-side encryption solutions.

2.1 The Key-Cascade

Our "Key-Cascade" key management mechanism is based on a tree structure in which each node contains encryption keys and child nodes are encrypted with a key from their parent [3, 10]. Figure 4 shows an example of this structure with Key 0 as the master key. Each inner node contains a list of encryption keys and each key is used to encrypt one of the node's children. E.g. in Figure 4, Key 1 is used to encrypt the child node containing Keys 17 through 21. Because this child node is encrypted with Key 1, it is called Node 1. The leaves of this tree are nodes containing encryption keys for the actual data objects. Each (encrypted) node is stored as an object inside the object store and identified by its node ID. The IDs for keys, nodes, and objects are used for accessing the Key-Cascade data structure. The IDs are assigned so that only an object ID is needed to calculate the list of node and key IDs along the path to this leaf. This allows decoupling the retrieval and processing of the encrypted nodes.

Figure 2 shows how cryptographic deletion is realized on this data structure. The purpose of the Key-Cascade is to transfer the property of secure deletion from the master key (stored in TPM) to the large number of object keys. This is achieved through the hierarchical dependency between the keys. Once a key becomes inaccessible, all its child nodes become inaccessible as well, leading to cryptographic deletion of the corresponding objects. Once the master key is securely deleted (by the TPM), all the nodes of the tree become inaccessible and all the objects become securely deleted as well.

We use the logarithmic height of the tree in order to cryptographically delete individual objects with minimal overhead: Figure 2 shows how Object 7 is deleted by generating a new (and deleting the old) master key and modifying a path of nodes. The nodes along the path from master key to Object 7 get copied and modified in two ways: i) internal nodes are copied while the key for the child node on this path is replaced by a newly generated one. ii) leaf nodes

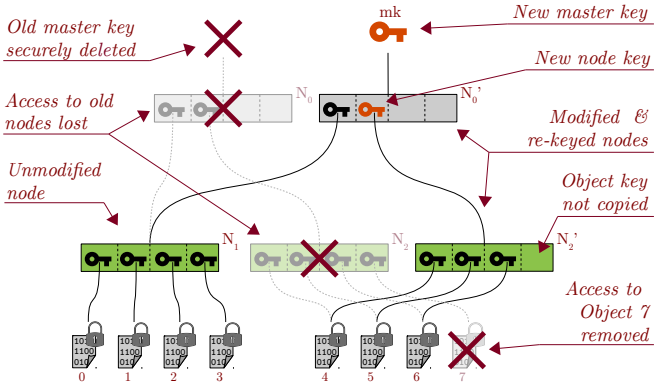

Figure 2: the Key-Cascade data structure after deletion of object 7.

contain the object keys. This node is copied as well but the object key (which should be deleted) is not copied. In both cases the copied nodes are then re-encrypted with their new parent key. All off-path branches remain unaffected and are now accessible through the new master key and modified nodes.

All the old nodes and objects become inaccessible because the old master key was securely deleted by the TPM. It is not possible to restore these deleted objects even if all the old nodes can be restored, because the old master key is no longer available. This recursive "re-keying" of nodes always includes replacing the master key. In order to reduce cost, deletions can be processed in batches. The master key then only needs to be replaced once per batch.

2.1.1 Key-Cascade properties

The properties of the Key-Cascade are determined by two parameters: The tree height h and the node size S_n. These properties include the maximum number of object keys, the number of nodes, and the size of the whole data structure. In the following, we give two examples for these properties. Each key has a size of 32 bytes in these calculations because we use the AES256 encryption algorithm.

Example 1:

Tree height $h = 2$, node size $S_n = 2^2 = 4$.

This results in a cascade consisting of 5 nodes with space for 16 object keys. When fully utilized, the Key-Cascade needs 640 bytes to store the nodes. Re-keying requires up to 12 operations. With data objects of 100 kilobytes average size, this cascade can store keys for 1.6 megabytes of data. The cascade therefore imposes a storage overhead of 0.04%

Example 2:

Tree height $h = 3$, node size $S_n = 2^8 = 256$.

This more realistic parameter setting results in a cascade consisting of 65,793 nodes with space for 16,777,216 object keys. When fully utilized, the Key-Cascade needs 514 megabytes to store the nodes. Re-keying requires up to 774 operations (note that these are small in-memory operations). With data objects of 100 kilobytes average size, this cascade can store keys for 1.6 terabytes of data. The cascade therefore imposes a storage overhead of 0.03%

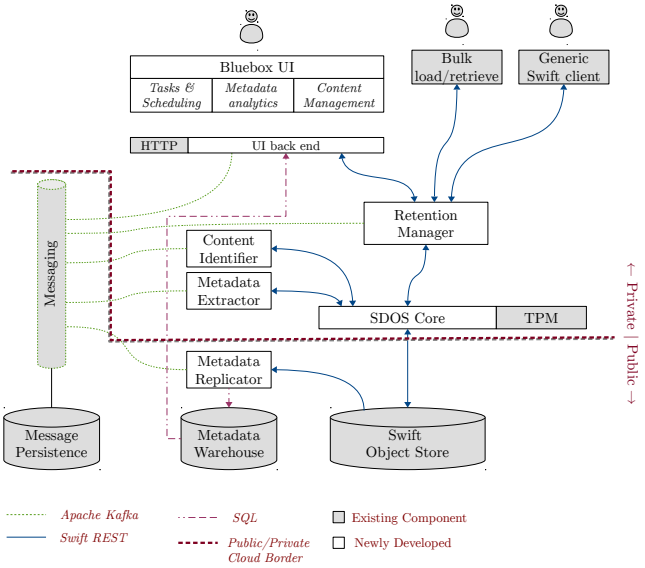

Figure 3: The architecture of the Micro Content Management system with hardware Trusted Platform Module (TPM).

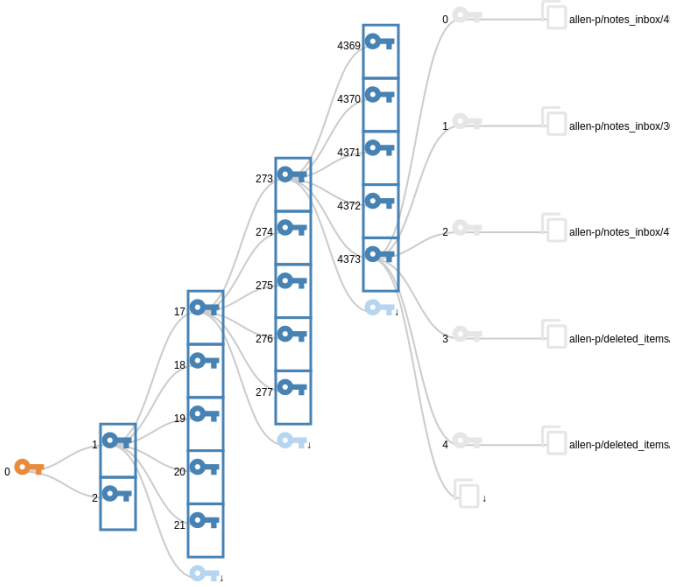

Figure 4: Screenshot of the interactive visualization for the key management data structure. Key 0 is secured by the TPM.

3. THE MICRO CONTENT MANAGEMENT SYSTEM (MCM)

In this demo, we present our proof-of-concept and bench marking application MCM[1]. MCM's functionality is based on on-premise Enterprise Content Management systems like IBM FileNet P8, but is designed to use outsourced cloud storage and client-side encryption. MCM stores objects and files inside storage containers in the Swift[2] object store. Whole containers can be transparently encrypted with a key-management mechanism that allows secure deletion of individual objects. MCM supports uploading and retrieving files, setting retention dates and scheduling deletion, extracting and viewing metadata, and analyzing and graphing analyses on this metadata. Our user interface also features interactive visualizations of the underlying key management data structures, which will be used in this Demo.

Figure 3 shows the high level architecture of MCM. The central component for this Demo is SDOS, the Secure Delete Object Store, which implements encryption as well as the key management for cryptographic deletion and also has the integration with the TPM.

We use three data management systems (bottom row of Figure 3): An Apache Kafka streaming platform for loosely coupled communication, an SQL database for storing and analyzing unencrypted metadata, and a Swift object store that holds all the encrypted data objects. Metadata is stored unencrypted in MCM because this allows us to execute queries on the cloud. The user can decide what metadata should be extracted from files, depending on their sensitivity. Cloud resources can then be used to query, search, filter, and analyze this metadata. File and container names are always stored as unencrypted metadata. If no further metadata is extracted, users must always retrieve (and decrypt) files before their content can be searched or analyzed locally.

[1]https://github.com/timwaizenegger/mcm-sdos
[2]http://docs.openstack.org/developer/swift/

We use the Swift REST-API protocol for our internal components in MCM, as shown by the blue lines in Figure 3. This protocol is used by the Swift object store and other large key/value stores (e.g. Ceph[3]) and their clients. Encryption and cryptographic deletion are handled by our SDOS component which is realized as such a Swift API-proxy. This enables us to use any unmodified Swift backend (e.g. SaaS) as well as any existing Swift clients. The API-proxies form a flexible pipeline. All MCM components can run multithreaded or distributed to enable horizontal scaling and high availability.

The Kafka streaming platform is used for triggering the execution of jobs for metadata extraction and replication as well as scheduled deletion of old objects. We use a relational database as a replicated metadata warehouse, as Swift lacks advanced querying capabilities for metadata (only retrieving and listing is possible). All the object metadata is primarily stored in Swift and then replicated to the RDBMS for analysis.

The location where the components from Figure 3 run is critical to the security of the system. In order to guarantee the secure deletion property, the content of the stored objects must never leave a trusted environment in unencrypted form. The same must be guaranteed for the encryption keys. Our SDOS encryption uses a tree structure for key management of which only the root key must be kept secure. All other keys are stored encrypted on Swift together with the data objects.

One possible separation of trusted/untrusted environment is given by the red line in Figure 3. It shows that all the data storage system can be outsourced to the public (untrusted) environment, because all sensitive data are encrypted. The master key for our key management mechanism is stored in, and never leaves the TPM.

[3]http://docs.ceph.com/docs/jewel/radosgw/swift

4. TRUSTED PLATFORM MODULES (TPM)

Trusted Platform Modules are a type of hardware security module that is used in PCs, Laptops, and Servers. They are already ubiquitous in those devices today and will achieve even higher spread in the future since Microsoft now lists a TPM as a mandatory hardware requirement for its Windows 10 operating system[4]. TPMs implement a specification by the Trusted Platform Group that defines core capabilities and security requirements [9]. TPMs generally contain a small storage as well as processing unit inside a tamper resistant physical package. Their most important feature is that certain areas of their storage unit can only be accessed by the internal processor. This means that some encryption keys can never leave the TPM but can only be used to de/encrypt data that is loaded into the TPM. TPMs furthermore have the capability to securely delete stored keys, and replace them with newly generated ones. This provides a secure basis for cryptographic deletion since a key is positively unrecoverable if it never leaves the TPM and is securely deleted inside the TPM later on.

The TPM's intended purpose is to support local disk and data encryption, as well as verified device identification. For data encryption, an encryption key is stored on disk but encrypted with the TPM master key. This encryption key can only be used after it was decrypted by an authenticated TPM. The actual data de- and encryption is then done by the main CPU, only de and encryption of the key is done inside the TPM. For device identification, TPMs contain an "endorsement key" that was signed by a trusted manufacturer master key. Remote services can challenge the TPM and verify the endorsement key in order to identify a certain machine. This is used for enterprise asset tracking as well as licence management for digital media (digital rights management).

TPMs can be used in custom applications with the limitation that no custom code can be run inside the TPM. Only the basic cryptographic operations are supported by the processor inside the TPM [11]. TPMs offer physical security and tamper resistance and can be used to secure master keys to custom cryptographic applications.

In MCM we use a TPM in order to store the master key on the SDOS core component. This master key (Key 0 in Figure 4) encrypts the first level of keys in a tree. The master key never leaves the TPM and is only used to en/decrypt the first level of the tree by loading this node into the TPM for processing.

5. DEMO OVERVIEW

This Demo will present a working prototype of a cloud storage system that offers transparent encryption with cryptographic deletion. We will show the theory behind our key-management mechanism (Key-Cascade), present the architecture of the cloud storage system, and demonstrate the integration with a Trusted Platform Module.

In our demo scenario we will first explain the layout of the system and the physical location of the individual components. We will then create new data containers with and without cryptographic deletion and show data ingestion and retrieval with different client applications. We will show how the encryption keys for new objects are generated and how they fit into the hierarchical Key-Cascade. We present the operations on the Key-Cascade including cryptographic deletion, show the capacity of the data structure as well as its scaling behavior. We then show how the Trusted Platform Module is integrated with the Key-Cascade and operations.

In this demo, the audience will learn about cryptographic deletion and its application to practical storage systems. Our integration of Trusted Platform Modules is relevant for applications outside of cryptographic deletion as well. Any system that employs cryptography can increase certain security aspects by integrating a TPM. Therefore, this demo is also relevant for researchers working in other areas of applied cryptography. Finally, our solution makes heavy use of the Swift object store and its REST-API which makes this demo relevant for researchers interested in Swift as well.

Screenshots of the user interface, all the application code, as well as more details about their capabilities, can be found on our Github page: https://github.com/timwaizenegger/mcm-bluebox

[4]https://msdn.microsoft.com/en-us/library/windows/hardware/dn915086(v=vs.85).aspx

6. REFERENCES

[1] Amazon glacier with vault lock, SEC 17a-4(f) & CFTC 1.31(b)-(c) compliance assessment. Technical report, Cohasset Associates Inc., Aug. 2015.

[2] Regulation (EU) 2016/679: General data protection regulation. Technical report, European Parliament and Council, 2016.

[3] J. Barney, D. Lebutsch, C. Mega, S. Schleipen, and T. Waizenegger. Deletion of content in digital storage systems, March 2016. US Patent 9,298,951.

[4] S. Beresford. Deletion of records from national police systems. Technical report, UK National Police Chiefs' Council, May 2015.

[5] D. Brown, C. Arend, and A. Venkatraman. EU data protection reform will drive growth in european security and storage markets. *IDC ESS02X*, Oct. 2015.

[6] T. Conde. To delete or not delete - that's the question: A company's obligations to preserve records under the new electronic discovery rules. Technical report, Stoel Rives LLP, 2009.

[7] S. M. Diesburg and A.-I. A. Wang. A survey of confidential data storage and deletion methods. *ACM Comput. Surv.*, 43(1):2:1–2:37, Dec. 2010.

[8] S. Garfinkel and A. Shelat. Remembrance of data passed: a study of disk sanitization practices. *Security Privacy, IEEE*, 1(1):17–27, Jan. 2003.

[9] Trusted Computing Group. TPM 1.2 protection profile, 2016.

[10] T. Waizenegger. Poster presentation: SDOS: Secure deletion in the swift object store. In C. Nikolaou and F. Leymann, editors, *Proceedings of the 9th Symposium and Summer School On Service-Oriented Computing*, volume RC25564 of *IBM Research Report*. IBM, Dec. 2015.

[11] V. J. Zimmer, S. R. Dasari, and S. P. Brogan. Tcg-based firmware, white paper by Intel corporation and IBM corporation trusted platforms, 2009.