EncSwift and Key Management: An Integrated Approach in an Industrial Setting

Enrico Bacis*, Marco Rosa*, Ali Sajjad†
*Università degli Studi di Bergamo, 24044 Dalmine - Italy Email: *firstname.lastname*@unibg.it
†British Telecom - UK Email: ali.sajjad@bt.com

***Abstract*—The use of cloud technology is continually expanding. Yet, in many scenarios the adoption of an external cloud service provider may be a worry for data confidentiality since it leads to a partially loss of control over data. One of the solutions for letting users put trust in a provider is the use of encryption to protect data. EncSwift [1] is a solution that provides transparent support for the encryption of objects stored on OpenStack based providers, adopting Barbican, the OpenStack secret storage, as a key manager. In this work we introduce a new key manager, BT KMS, already adopted in industrial systems, that offers a large set of features, and that it is designed to be flexible, transparent, and scalable. Moreover, we analyze the possibility of integration between the BT KMS and the EncSwift approach, and provide an architectural overview of this new integrated system.**

I. Introduction

A clear trend of users and organizations for storing data is moving them to the cloud. This provides great advantages, mostly in terms of economic cost and availability, but also introduces major concerns with regard to confidentiality. This becomes pivotal when considering the cloud provider as hones-but-curious, i.e., it behaves as expected but it cannot be trusted for accessing the content of data it stores. An efficient approach for dealing with an hones-but-curious provider is the use of encryption for protecting data before outsourcing them to an external cloud provider [1], [2]. In this way, data are protected both against privacy interfering providers and adversaries who may exploit vulnerabilities of the provider in order to gain access to the physical representation on disks. Moreover, the use of encryption offers the opportunity to effectively enforce access control by encrypting data with different keys, each of them shared with users according to an authorization policy [3], [4]. *EncSwift* [1] is a tool for protecting data confidentiality in OpenStack Swift, it uses different encryption keys to protect resources, according to the policies ruling access on them, and providing in this way fine graned access control.

In a dynamic system with frequent policy updates, the use of encryption brings not negligible complexity since keys must be shared and revoked in an efficient and trusted way. In fact, as though granting access to a resource only requires to share its encryption key, access revocation is a more relevant problem, because the revoked user could have stored local copies of the keys and, if the access control enforced by the server is faulty, she potentially could still access the plain content of the resource. Nevertheless, forcing the owner of the resource to download and re-encrypt objects with a fresh key would add a unbearable overhead to the performance of the system. In this case, Over-Encryption [5] offers a solution, adopted also in the EncSwift tool, that can manage access revocation in an efficient way, applying a server-side encryption layer such that this new encryption key, generated by the server, cannot be known to revoked users. Over-Encryption would provide a clear benefit both in terms of performance and security, but requires an efficient key management service able to manage a potential large number of encryption keys.

978-1-5386-0683-4/17/$31.00 ©2017 IEEE

The purpose of this work is to introduce a new efficient key management service, BT KMS, and investigate its integration with EncSwift, in a scenario with frequent policy updates, and thus with the generation and administration of a large number of encryption keys.

The remainder of this paper is organized as follows. Section II describes some basic concepts on the OpenStack infrastructure, together with the principles of EncSwift, focusing on one of its implementations. Section III describes the architecture of BT KMS and its integration with EncSwift. Section IV discusses related work. Finally, Section V presents our conclusions.

II. EncSwift

This section introduces the EncSwift approach. In Section II-A we briefly describe the OpenStack organization and its infrastructure. In section II-B we describe EncSwift principles for data confidentiality protection, and in section II-C we analyze how the Over-Encryption approach can be introduced into EncSwift for managing policy updates.

A. OpenStack Swift

In this paper, we assume that the user wants to outsource her data to a cloud provider based on OpenStack [6]. OpenStack, originally developed by Nasa and Rackspace has become today the *standard-de-facto* for private cloud. It offers a combination of open source tools for managing the core cloud-computing services of compute, networking, storage, identity, and image services, and more other projects can be bundled together for customizing the platform based on need. In this paper we focus on Swift, the object storage of OpenStack. Swift is designed to efficiently, safely, and cheaply store files, that can be accessed through a *tenant/container/object* structure. *Objects* correspond to files and are organized in *containers*, i.e., folders, whereas at the highest level of this hierarchy are

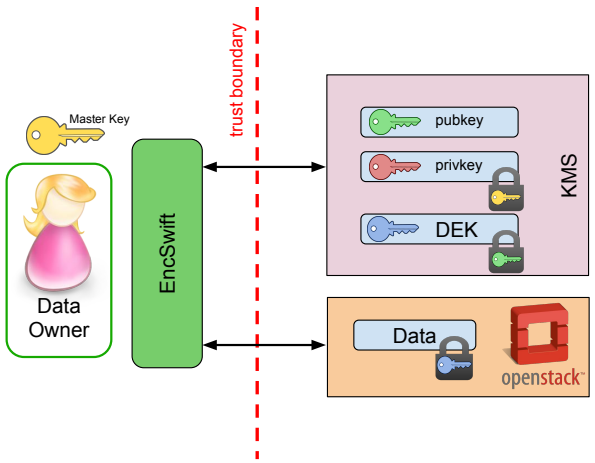

Fig. 1. EncSwift architecture

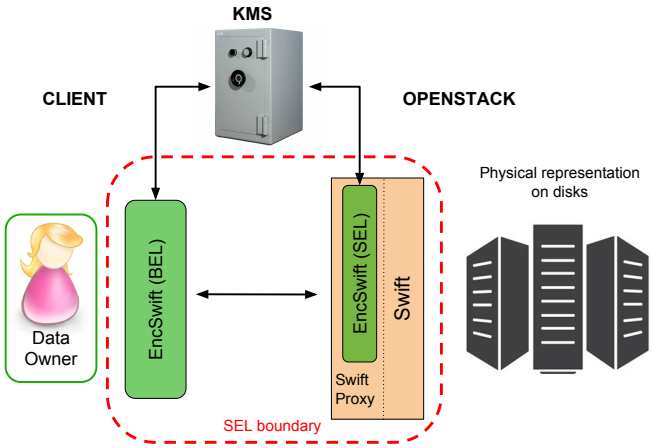

Fig. 2. Over-Encryption management in EncSwift

the *tenants*, sets of containers that are usually assigned to an organization. Swift also defines two levels of Access Control Lists (ACLs): *tenant* and *container*. A tenant-level ACL defines users that can perform administrative operations on the tenant. A container-level ACL defines $read|write|listing$ permissions on the container. There is no ACL associated with objects (the ACL of a container applies to all objects in it).

With regard to its internal structure, Swift it based on *WSGI* (Web Server Gateway Interface). This permits to define a *pipeline*, i.e., a chain of modules (the *middlewares*) that can enrich requests before they reach the main web server component (the Proxy Node), and that can modify the responses coming from it.

B. Protection of data confidentiality

EncSwift is a solution presented in [1] for enforcing data confidentiality and efficient access control in OpenStack Swift. EncSwift behaves as a transparent proxy, encrypting objects, with a symmetric key (DEK, *Data Encryption Key*) before uploading them to the cloud provider, and decrypting them at download time. Each DEK is associated with a container, such that all the objects stored in the same container are encrypted with the same DEK, that must be shared along all the users that can access it. Every user is also associated with a signature key pair (for signing messages), and a RSA key pair, that can be used to asymmetrically encrypt the DEKs she owns, in order that she can store them in an encrypted form, i.e., as KEKs (*Key Encryption Keys*) Owing to the fact that users may want to access their data from several devices, a secret known only to the user is needed (MK, *Master Key*). All the other keys (both the RSA key pair and the KEKs) are stored in a key management service. The Master Key is therefore used to encrypt the RSA and signature keys of the user. OpenStack already offers a secret storage, Barbican. Barbican stores secrets, that are equivalent to objects in Swift, and that can be organized in containers, just like Swift. Each Swift container then has a corresponding container in Barbican, storing both users' RSA and signature key pairs, and KEKs. Note that keys are stored in encrypted form in Barbican since we do not trust the cloud provider for accessing the sensitive content of outsourced objects. Indeed, storing keys in plaintext in a Barbican container would put the confidentiality of outsourced data at risk. Even if keys are stored in encrypted form at the cloud provider, in [1] the authors decided to use Barbican for their storage, in contrast to Swift, because it enforces additional security measures against outside attacks. EncSwift architecture is shown in Figure 1.

C. Policy Update

Due to the adoption of policy-based encryption, a modification to the ACL of a container spreads to the encryption policy of that container. In fact, a policy update would require to change the encryption key for that container, but this is unfeasible since a user would be required to download the whole content of a container, re-encrypt it and re-upload everything. Over-Encryption [5] is the solution that EncSwift adopts in order to prevent this excessively onerous operation. The integration of Over-Encryption requires the addition of a layer of encryption. Indeed, a first layer of encryption (BEL, *Base Encryption Layer*) is applied at client side in order to provide data confidentiality against an honest-but-curious provider, enforcing the initial access control policy. A second layer of encryption (SEL, *Surface Encryption Layer*) is then applied server side in order to enforce policy updates. Policy updates can be of two kinds: grant and revoke. In case of a grant access to a container, the data owner has to share the encryption key used for protecting resources at the BEL level, i.e., the BEL DEK, with the new user, creating a new KEK and sharing it through the Key Management Service (KMS). In case of revoke, the objects in the container must be protected at the SEL level with a new DEK, generated by the server. In [1] three implementations for the Over-Encryption are illustrated. In this paper we focus on the *on-the-fly* mode. This means that Over-Encryption is enforced every time a user downloads an object from that container, by generating a new SEL DEK to encrypt the resource on-the-fly. The Over-

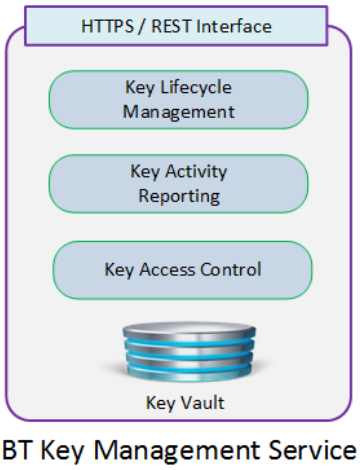

Fig. 3. Architecture of the BT KMS

Encryption can be easily integrated in EncSwift by inserting a new custom middleware in the Swift pipeline, thus it requires a modification of the server. The SEL DEK permits to enforce a fine grained access control mechanism at container level. It is generated by the custom middleware at the first access to the container after a policy update, and stored as a KEK for every authorized user in the KMS. Figure 2 shows the architecture of a policy update.

The number of keys that Barbican must manage increases exponentially with the number of policy updates. This is the rationale behind the investigation on the possibility of integration of EncSwift with a new efficient KMS.

III. Industrial Key Management for EncSwift

This section introduces the main contributions of our work. In Section III-A we describe a new industrial key management service, BT KMS, together with its architecture. In Section III-B we analyze a possible integration between this industrial key manager and the EncSwift approach for protecting data confidentiality and enforcing access control.

A. BT Key Management Service

The main purpose of the BT Key Management Service (KMS) is to allow users to securely manage their encryption keys and certificates securely, either in a cloud based or in an on-premise environment. Furthermore, in a cloud based setting, the BT KMS can be provisioned and managed via the BT Service Store [7], which enables its users to create isolated and compartmentalized key management domains. This allows the users to manage their keys for use in multiple cloud platforms in a secure multi-tenant environment. The main components of the BT KMS are shown in Figure 3 and are described in more detail below.

- *HTTPS/REST Interface*: it offers an intuitive web-based management console for enterprise-scale administration, as well as Single Sign-On capabilities that can be seamlessly integrated with LDAP based user identity management services. It also supports management through the use of a full-featured REST based API.
- *Key Lifecycle Management*: it provides complete lifecycle management of the keys, including key creation, storage, import, export, rotation, revocation and deletion. Thus users have complete control over every aspect of a key during its existence. It also supports multiple API standards like PKCS#11, Microsoft Extensible Key Management (EKM) and OASIS KMIP, which helps with the automation of various operational tasks.
- *Key Activity Reporting*: it implements the ability to audit and report on all activities relating to keys, so that the users are able to generate comprehensive and granular audit logs of encryption key and certificate management activities. This is an essential requirement for a lot of compliance standards, and BT KMS complies with the FIPS 140-2 Level 1 standard.
- *Key Access Control*: lastly, it is tightly integrated with an access control capability that governs the rules and policies for releasing the relevant data encryption keys to the authorized users and processes. It also enforces the storage and retrieval to and from the key vault, which provides high availability storage and backup of the keys.

One of the core requirements from users of cloud based services is maximum and transparent control and ownership of their data in the cloud eco-system. This can be realized by utilizing the BT KMS, as it enforces the user-based management of the cryptographic keys, so that only the users are able to authorize policy-based release of keys to trusted applications. Even the cloud service provider on which the BT KMS is deployed has no view or control of the users' keys and other security credentials. This approach also keeps the keys separate from the data that they are protecting.

Each instance of the BT KMS contains a public/private key pair (RSA key pair), which is used to protect the Key Encrypting Keys (KEK) and other sensitive security objects stored within the Key Vault. The RSA key pair can be changed periodically without any impact to the keys and credentials being protected by it. A KEK is a random AES-256 key which is used to protect the Data Encryption Keys (DEK) while they are at-rest in the Key Vault or in transit from the BT KMS to the client application. The DEKs are AES-256 keys that are used for the actual encryption of the objects, and are always encrypted at-rest and in transit. This minimizes the exposure of the DEK, as well as the need for its frequent rotation.

B. Integration between BT KMS and EncSwift

We decided to analyze possible challenges and enhancements to the EncSwift solution in case of adoption of the BT KMS in place of Barbican. Obviously, a research tool as EncSwift would benefit from the integration of a full real industrial key management service. Indeed, BT KMS is designed to bring good performance and to be scalable in a real industrial scenario, and thanks to its flexibility and the wide number of communication protocols it supports, BT KMS would fully fit into the EncSwift architecture. Nevertheless,

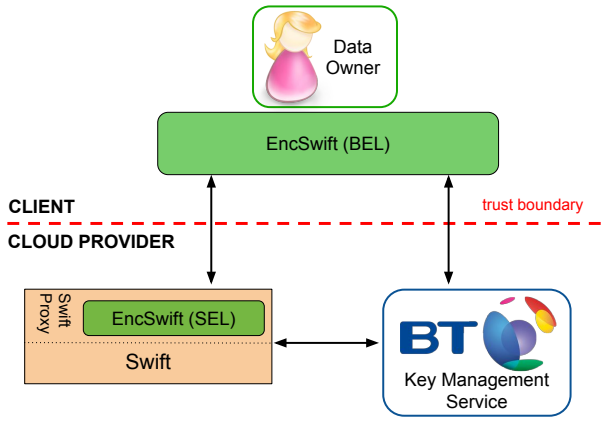

Fig. 4. Architecture of EncSwift and BT KMS integration

this integration would require a low implementation effort in order to fix some differences in the key usage. First of all, both BT KMS and EncSwift requires to store KEKs. Yet, in EncSwift KEKs are produced as symmetric DEKs asymmetrically encrypted with users' RSA keys, on the contrary BT KMS supports encryption of DEKs with a symmetric KEKs, that are encrypted themselves with users' RSA keys. A possible integration of the two architectures would require to choose one of the key treatment alternatives.

In our analysis, we adopt the EncSwift key structure, and deprecate the use of BT KMS encrypting keys, i.e., DEKs are asymmetrically encrypted with users' RSA keys. An architectural overview of the integration is shown in Figure 4. EncSwift still relies on an OpenStack Swift object storage, but all the encryption keys are stored at the BT KMS instead of Barbican. Moreover, the EncSwift SEL middleware implemented to enforce Over-Encryption can communicate with BT KMS in order to store and retrieve the SEL DEK.

IV. Related Work

A clear trend in the last years has been the increasing role of cloud computing, for both computation and storage. In this paper we focused on EncSwift [1], a solution for protecting data confidentiality and enforcing access control through the integration of Over-Encryption technique [5], adapted to operate in the OpenStack Swift scenario [8]. The original Over-Encryption proposal assumed the presence of a single data owner instead of a scenario with resource sharing, and operated with an abstract provider capable of resources and keys management. An extension of the proposal in [3] to a multi-owner scenario has been presented in [4] and is based on a Diffie-Hellman scheme, but it still considers an abstract provider.

Being one of the most widespread modern cloud infrastructures, OpenStack has recently been in the spotlight. Much research focus on the role of Swift, e.g., in [9] the authors propose a framework to encrypt objects, relying on an external server to be used as a KMS, but do not address the problem of efficiently enforcing access revocation. In [10], the problem of integrating a new external key manager into OpenStack is dealt with, but it still relies on Barbican.

V. Conclusions

This paper introduced a new solution for key management services, BT KMS, developed by BT and already adopted in real industrial settings. Thanks to its flexibility and well structured architecture, we investigated the possibility of a new integration between it and the EncSwift approach [1], which implements a solution for protecting data stored in OpenStack Swift. EncSwift guarantees data confidentiality and integrity, also naturally regulating - via encryption - access to them. With frequent policy updates, there is a trend to generate a large number of encryption keys, that EncSwift stores in Barbican, OpenStack secret storage. In our investigation, we were able to build an architecture of the new integrated system that could be produced with a low implementation effort.

VI. Acknowledgment

This work was supported by the EC within the H2020 under grant agreement 644579 (ESCUDO-CLOUD).

References

[1] E. Bacis, S. De Capitani di Vimercati, S. Foresti, D. Guttadoro, S. Paraboschi, M. Rosa, P. Samarati, and A. Saullo, "Managing data sharing in OpenStack Swift with Over-Encryption," in *Proc. of the 3rd ACM Workshop on Information Sharing and Collaborative Security (WISCS 2016)*, Vienna, Austria, October 2016.

[2] W. Wang, Z. Li, R. Owens, and B. Bhargava, "Secure and efficient access to outsourced data," in *Proc. of the 2009 ACM Workshop on Cloud Computing Security (CCSW 2009)*, Chicago, IL, USA, November 2009.

[3] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Encryption policies for regulating access to outsourced data," *ACM Transactions on Database Systems (TODS)*, vol. 35, no. 2, pp. 1–46, April 2010.

[4] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Pelosi, and P. Samarati, "Encryption-based policy enforcement for cloud storage," in *Proc. of the 1st ICDCS Workshop on Security and Privacy in Cloud Computing (SPCC 2010)*, Genova, Italy, June 2010.

[5] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Over-encryption: Management of access control evolution on outsourced data," in *Proc. of the 33rd International Conference on Very Large Data Bases (VLDB 2007)*, Vienna, Austria, September 2007.

[6] "OpenStack Project," http://www.openstack.org.

[7] G. Ducatel, J. Daniel, T. Dimitrakos, F. A. El-Moussa, R. Rowlingson, and A. Sajjad, "Managed security service distribution model," in *Proc. of the 4th International Conference on Cloud Computing and Intelligence Systems (CCIS 2016)*, Beijing, China, August 2016.

[8] E. Bacis, S. D. C. di Vimercati, S. Foresti, S. Paraboschi, M. Rosa, and P. Samarati, "Access control management for secure cloud storage," in *Proc. of the 12th International Conference on Security and Privacy in Communication Networks (SecureComm 2016)*, Guangzhou, China, October 2016.

[9] H. Albaroodi, S. Manickam, and M. Anbar, "A proposed framework for outsourcing and secure encrypted data on OpenStack object storage (Swift)," *Journal of Computer Science*, vol. 11, no. 3, pp. 590–597, 2015.

[10] D. Sitaram, S. Harwalkar, U. Simha, S. Iyer, and S. Jha, "Standards based integration of advanced key management capabilities with OpenStack," in *Proc. of the 2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, Bangalore, India, March 2015.