Journal of Computer Science

Research Notes

A Proposed Framework for Outsourcing and Secure Encrypted Data on OpenStack Object Storage (Swift)

Hala Albaroodi, Selvakumar Manickam and Mohammed Anbar

National Advanced IPv6 Centre (NAv6), Universiti Sains Malaysia, 11800, Penang, Malaysia

Article history
Received: 09-02-2015
Revised: 11-03-2015
Accepted: 28-04-2015

Corresponding Author:
Hala Albaroodi
National Advanced IPv6 Centre (NAv6), Universiti Sains Malaysia, 11800, Penang, Malaysia
Email: hala@nav6.org

Abstract: Despite the numerous potential benefits of Open Source Cloud Computing (OSCC) in several industrial and academic-oriented environments, OSCC could be also associated with some risks. However, which a proper awareness to the cloud consumers or organisations, these risks can be clearly identify and avoided. OpenStack Swift security can provide a greater understanding of how OpenStack Swift functions and what types of security issues arise therein. In this study, a lightweight and robust cloud-based security model for OpenStack object storage within a cloud computing environment is proposed. Swift is a multi-user based model in which every owner encrypts her/his files; each owner uses different levels of cryptographic security. A reduction in the key distribution complexity in this diverse model with a variety of security based settings is critical. Note that proposed model incorporates cryptographic algorithms at the first level (authentication/authorisation) and a hash function to introduce a more secure access method for authentication and authorisation.

Keywords: Security, Cloud Computing, Cloud Storage, OpenStack, OpenStack Object Storage Swift

Introduction

Recently, a strong interaction between cloud computing and Open Source Software (OSS) has been demonstrated by the announcement of several Open Source Cloud Computing (OSCC) projects, such as OpenStack, OpenNebula, Hadoop and CloudStack (Rodriguez-Martinez *et al*., 2010; Hala *et al*., 2013; Albaroodi *et al*., 2013).

Typically, the security of data in the could computing service providers would be the first matter to the most of the potential cloud customers. In fact, questioning around the security of personal data is one of the ultimate rights to customers before making any decisions. It is known that any incident related to the security and privacy of consumers has the capability to break the brand equity and the consumer trust. Further, the level of privacy and the security models which implemented by different cloud platforms reflect the acceptance of the developments of several applications. Thus, there will be always an open door to investigate security regardless of the innovations of new technologies such as cloud computing. Investigating the security of cloud computing will be of great importance when the open source software OSS technology comes over the picture Armbrust *et al*., 2010).

The security concerns of cloud computing is investigated in this study as the primary motivation of this research and a user-trusted model to mitigate and avoid these concerns is proposed with the aid of OSS (Popovic and Hocenski, 2010; Okuhara *et al*., 2010; Cooper, 2013). As it forms one of the major parts of this research, the object storage components of OpenStack is considered in this study. However, other concerns related to other components of OpenStack are not considered. Moreover, as has been identified in the security issues of this OSS-based solution, two main security areas are investigated and solved: The identity and access management and the access control and data protection (Baset *et al*., 2013; Khan *et al*., 2011; Kim *et al*., 2013). Further, in order to narrow the scope of literatures concerning the security of the object storage components of OpenStack, only those literatures targeting the areas of identity and access management, access control and privacy and data protection are selected (Chou, 2013).

Security Issues in OpenStack

As data needs to be immediately available and stored indefinitely on a variety of devices, the demand on storage is rapidly changing. This demand requires the construction of storage silos that utilize non-web protocols, which are restricted to specific applications. Online video, social media, user-uploaded content, gaming and SaaS applications are some of the demands that are driving this change.

Public cloud storage services have strived to satisfy these new storage needs; however, all organisations are not capable of or should be allowed to use public cloud storage. To provide these changing needs, a storage model must be able to handle web-scale workloads with many simultaneous readers and writers to a data store.

The possibility of utilising cloud computing based on the OSS technology such as OpenStack, is promising as it is considered to be pioneer services of OSCC. As one of the two basic components of the OpenStack project, Swift is employed to satisfy a variety of demands. Swift's usage includes small deployments for storing VM images, mission-critical storage clusters for high-volume websites, custom file-sharing applications, mobile application development, data analytics and private storage IaaS. Swift is OSS under the Apache 2 licenses and has over 70 contributors; new developers are contributing every year.

However, weaknesses in hardware, networks, software and service compel researchers to explore security issues. Therefore, interesting in security issues OpenStack object storage environments as illustrates in Fig. 1. The most common security threats on the OpenStack Swift environment are as follows (Gellman, 2012). The following subsections verify these challenges to OpenStack object storage:

The cloud computing paradigm has attracted considerable attention as it provides Infrastructure as a Service (IaaS); IaaS providers can enjoy virtually infinite storage and cloud computing resources. Willing providers are increasingly shifting their data storage and application services into cloud computing, which decreases their operational costs, rather than building specialised data centres. Data storage services in the cloud computing environment should be available to everyone but many security risks exist that may delay its adoption (Gansen *et al*., 2010; Venkatesa and Palaniswami, 2012). The main issue concerns the security of a user's sensitive data and who can obtain access to storage when data are saved in a cloud computing server. Users lose physical control over their sensitive data when they directly place these data in the control of a server that cannot provide privacy assurance (Venkatesa and Palaniswami, 2012; Dai Yuefa *et al*., 2009; Albaroodi *et al*., 2014a; Ponnuramu and Tamilselvan, 2012).

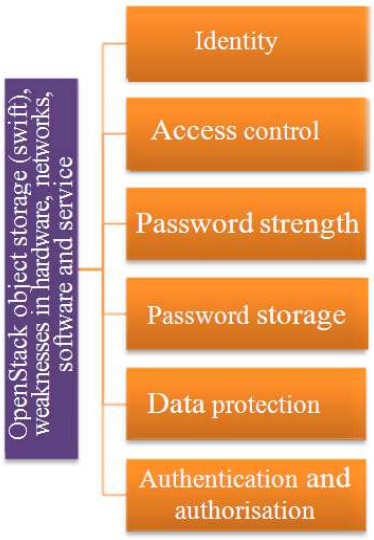

Fig. 1. OpenStack swift weaknesses

Table 1. Summary of the core related work

Security criteria	Shortcoming
Identity	OpenStack identity of data user's from client to storage server is not secure. In addition, high-value sensitive information may leak from data storage by vulnerability.
Access control	Access control (authentication) must be designed to prevent accidental users from engaging in malicious practices and spammers and bots from registering on any site.
Password storage	The URL of the file that is used to save user information must be examined prior to making any changes to rights of access. Only the Swift user should have access to this file to prevent other users from obtaining user credential information. Users should be registered with appropriate passwords that contain special characters.
Securing password	The strength of passwords should be checked before registering users can reject weak passwords. In addition, these passwords should be hashed before they are saved in the model. Before saving, strong passwords should be concatenated to a username and hashed with an appropriate algorithm. The SHA-256 algorithm can be employed to hash the passwords.
Data protection	All important and sensitive files should be encrypted before they are uploaded to OpenStack Swift to prevent users from viewing files that belong to other users.
Authentication and authorization	User authentication and authorisation functionality in an OpenStack Swift environment utilise non secured process during the account generation and the keystone where the account save not protected.

Thus, this research has focused on the security variances within the OpenStack object storage (Swift) environment. Particularly, several works which are addressing the security concern of the swift component of OpenStack are discussed. Generally and as has been individually mentioned in the presentation of each work, enhancing the security mechanism of swift has lacked to provide comprehensiveness (Hala *et al*., 2013; Albaroodi *et al*., 2014a; 2014b). These lacks were identified to by ranging from securing the identity, access control, passwords, protecting the data, to the protecting of the authentication and authorization process. In details, an explanation to the technical shortcomings of previous works with regards to the above criteria is given in Table1.

Objectives

The main goal of this research is propose a security model for OpenStack Storage to address the shortcoming security issues in OpenStack Swift this research are as follows:

- To achieve strict authentication and authorisation by designing implications for the use of CAPTCHA, this ensures that only appropriate users have access
- To maintain the safety of usernames and passwords by a symmetric algorithm and hash function before usernames and passwords are saved in the authentication module of proposed model and verify their robustness by cryptanalysis
- To establish a trust relationship among entities as a vital aspect of the creation of a security mechanism in the proposed model environments by the keystone module to authorise identity federation and prevent being locked into proprietary solutions
- To centralise the resource information of an available proxy using the keystone module
- To protect sensitive files using an asymmetric algorithm prior to uploading or downloading these files to prevent users from viewing files that belong to other users. To check the vulnerability by applying cryptanalysis on the asymmetric algorithm. To address the fault tolerance architecture, proposed model backup database repository. In addition, the loud balancing will be solved by multiple proxy servers

Existing OpenStack Swift

OpenStack is aIaaS delivery model that can be distributed in any deployment model. When OpenStack is deployed in a private cloud delivery model, a cloud vendor maintains security control from layer 1 to layer 7. When OpenStack is deployed in public and hybrid clouds; subsequently, all security services are available through web services at the previously discussed application levels. Figure 2 depicts the OpenStack Swift authentication.

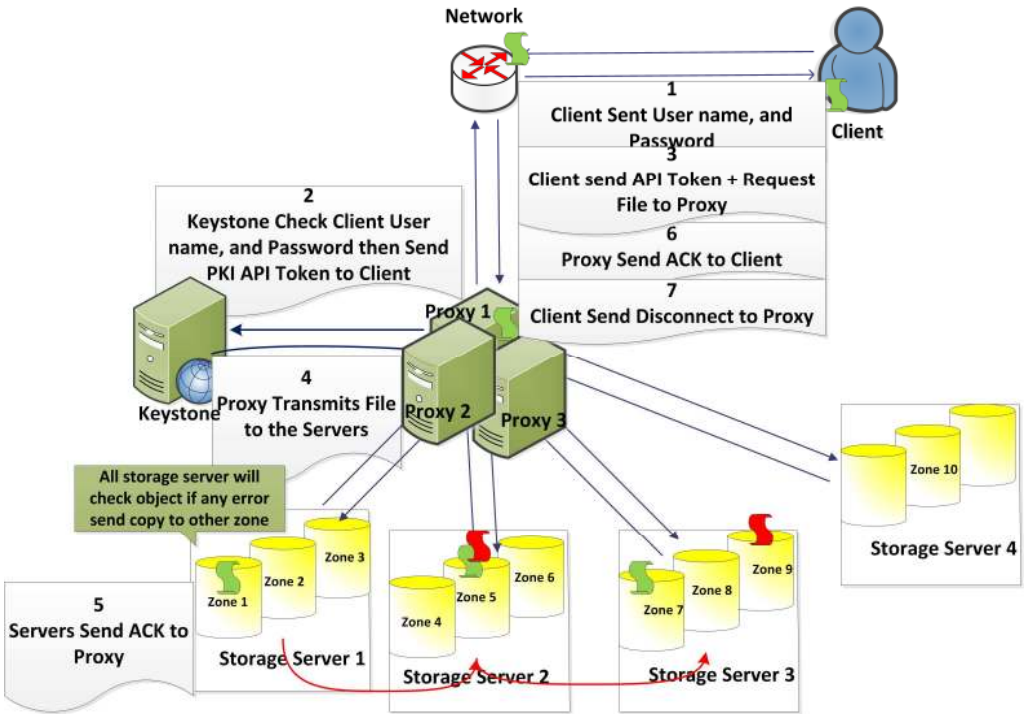

Fig. 2. OpenStack Swift (AS, 2010)

Based on the discussion in chapter two identifying the problem integrated with the current OpenStack Swift, which uses Keystone as a principal for user authentication security (Adam.Younglogic.Com, 2012). The authentication process is based on usernames and passwords, which are transmitted as clear text. A successful authentication on a storage service (Swift) must invoke the Keystone server, which generates a valid token for use by the end-user for additional processing. The use HTTP protocol in all communications (secure banking online transaction system, secure cloud-based data services etc.) is considered to be less secure than the use of HTTPS. HTTP increases the potential for particular security issues, such as access identity. The randomness of the 32-bit token remains ambiguous compared with a signed token by a certificate authority.

These tokens are short-lived as they are valid for only 24 h; users have to authenticate themselves again after the validity of the token expires. With the implementation of a Public Key Infrastructure (PKI), Keystone can provide a Strong authentication Key Appliance that uses a third-party library to develop an enterprise key management infrastructure, which supports the services of PKI and provides symmetric key management libraries. However, this library does not include features that can securely manage keys at the cloud platform. This library does not include features that can securely manage keys at the cloud platform. It requires a separate server for key storage and a compromise of this server can create a bottleneck for key security and generate the certificate for longer time validity. OpenStack has a greater number of weaknesses regarding data protection (Cooper, 2013; Cigoj and Klobučar, 2012). In this context, primarily focus on these enhancements, which consider security as an important key metric for all levels of OpenStack Swift.

Proposed Framework

This research focused on OpenStack security issues. Specific component is Swift of OpenStack is considered to be secure, whereas other components like nova, cinder and dashboard and so on need to be improved. OpenStack not supports password complexity requirements and passwords are stored in a plaintext text format and some. The access on sensitive data files not secure can be attacked. Information transformed within the cloud is not protected via using encryption and decryption files techniques.

The present research framework is limited to the development and evaluation of a new model in which enhanced security will be applied in the form of symmetric and asymmetric cryptographic algorithms; proposed model consists of four main modules: (1) user credentials, (2) keystone module, (3) proxy module and (4) Database module. The user credential module addresses secure information and the Keystone module handles authentication, authorisation and scheduling. The objective of the proxy module is to manage public and private keys using asymmetric encryption. The last module is a database module that is responsible for storing and retrieving data. These modules will be detailed in the next subsection. Fig. 3 shows the proposed model framework.

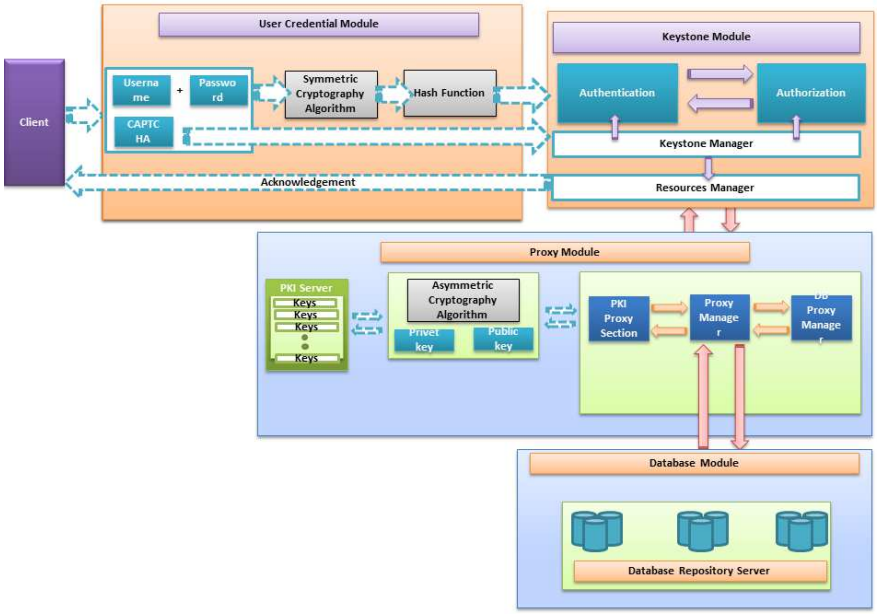

Fig. 3. A framework for proposed model

Module I: User credentials

In authentication, the new user will create the credential by a sign-up that will be approved by the model administrator. After the user credentials have been successfully created, the user will be given a username and password.

To prevent possible robot and insecure attempts of authentication, the CAPTCHA functionality is introduced and displayed for the user during the authentication process. After the user has entered his credentials, the authentication information, namely, username, password and CAPTCHA, will be encrypted using a symmetric cryptography algorithm and the hash function is applied to improve the security of the process. The secure information is sent to the authentication that will check the hash value and then apply the symmetric cryptography algorithm to verity a possible attempt of eavesdropping on the data. Once satisfied, the validity of the username, password and CAPTCHA with be verified.

Module II: Keystone Module

After the Keystone module is executed, it will wait for the client request. The process will be repeated in the case of no response; otherwise, the user credentials will be decrypted and the hash function will be applied. With a successful decryption and hash function, the username and password will be verified. On the other words, when verification is successful:

- If user credential is correct, then
- Authorization will be checked
- Else
- A failure message will be sent back to the client.
- End if

In terms of validation of authorization type:

- IF user type is a “normal user” THEN
- Model request sent to keystone resources manager
- Process will be continued
- Else IF user type is “administrator user” THEN
- Display
- Administrator panel
- Display User credentials
- Wait for user response.
- If the response key pressed is “exit” THEN
- Display Success message and exit
- ELSE IF response key pressed is “add/delete/modify” THEN
- Display User credential screen
- End IF
- END IF

Module III: Proxy Module

The proxy module communicates with the PKI server that stores the private key. The least busy server that is connected with the proxy module will handle the private and public keys. Asymmetric encryption will be utilised in this part to increase security. Symmetric encryption exhibits a weakness as the secret (which can be a number, word, or string of random letters) key can be easily accessible by everyone.

Module IV: Database Module

The database module is a crucial modules of proposed model in which the user resources will be saved securely. The information stored in the database repository is in the form of text and multimedia (audio and video). Per the request of a user, the following information will be sent and received by the user. To address the fault tolerance architecture, proposed model contains a backup database repository. If there is user congestion or fault in the main database, the user can access the backup database repository. The database module contains the database repository server. The database repository server is responsible for saving and retrieving data securely that is related to the entire model.

The Implementation Details of Proposed Model

The graphical interface of proposed model will display the username and password and prompt a client to enter his credentials. After the user credentials are entered, the process is initiated by displaying CAPTCHA on the screen to verify the validity of the remote user. After the correct CAPTCHA has been entered, the data are encrypted and a hash value is applied prior to submitting to the keystone authentication server.

Note that the CAPTCHA value will be sent to the client (hidden to the client) and verified on the client’s machine to reduce the cost of communication between the client and the server. After the CAPTCHA value is authenticated on the user’s machine, the data is transmitted to the keystone authentication server. If the keystone authentication is false, the wrong user credential message will be transmitted from the keystone authentication server to the client, where the user must re-enter a username and password. After correct user authentication then the user authorisation will be verified to determine whether the type of user is a normal user or an administrator (admin) user. In the next phase, the keystone manager will verify the proxy server credential information, such as hardware processing power (CPU), the available memory resources (RAM) and the number of active users in its database. The database will be updated periodically to obtain current proxy server credential information.

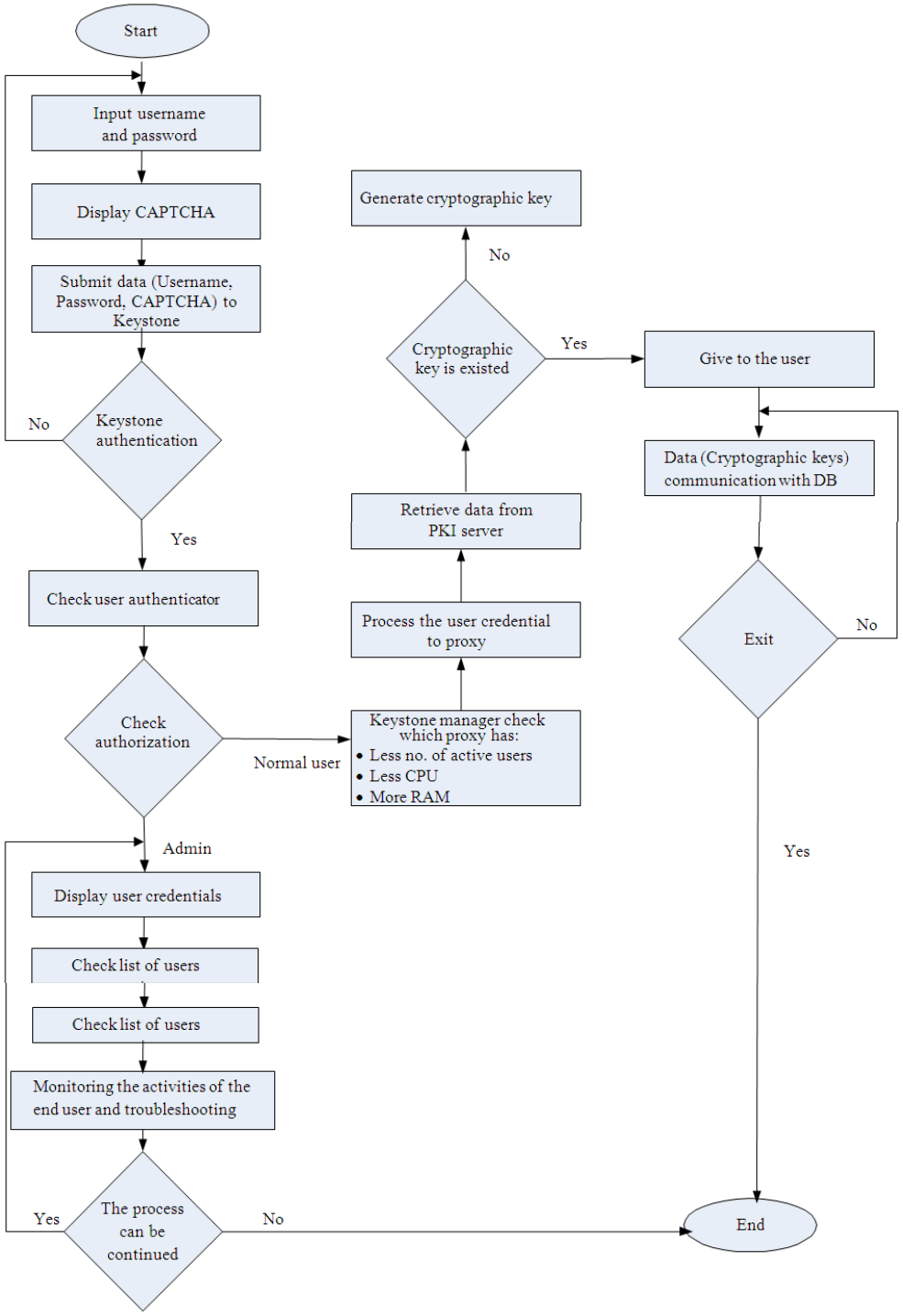

Fig. 4. Proposed model flow diagram

The Keystone manager will select the least busy proxy server and send its credential information to the client to ensure that it will be redirected to the specified proxy server for communication. The process is depicts in Fig. 4.

After the user receives the proxy credential information from the keystone manager, it will directly connect to the proxy server. The communication is initiated with the PKI in which the RSA keys will be retrieved from the proxy database and sent to the client.

When the client receives the encrypted RSA key, the secure communication will be initiated between the user client and the proxy server. After the key is provided and any interruption occurs via the communication, a new key is generated.

Data will be retrieved from the database module and the process will end if the user wants to exit; otherwise, communication with the database will continue. If the administrator is using the model, the user credentials will be displayed. The model administrator has the authority to check the list of users, troubleshoot any problems in the model and monitor the activities of the end user for the purpose of security.

Characteristics of the Proposed Model Vs. the OpenStack Swift

A characteristics comparison between the proposed model and the OpenStack Swift architecture is provides in Table 2. The first feature is the authentication of a user prior to accessing the entire model in the OpenStack Swift model only uses tokens identity per user supplement the negotiation protocols within the Keystone; Even, the encrypted password login not enough which can be compromised by eavesdropping. Proposed model is more secure because the authentication feature utilises symmetric cryptography (Blowfish algorithm) with the MD5 hash function and the CAPTCHA feature for greater security. Second feature is an authorisation, proposed model provides user-level authorisation with two types of users, namely, a normal user and an administrator (admin) user, the latter being granted a greater number of privileges if compared with OpenStack Swift admin because in proposed model the admin is customizable. Third feature is keystone manager, to verify the least busy proxy to offer a faster service to the user. This feature is not enabling in the OpenStack Swift.

Forth feature PKI, this feature is provides a strong secure appliance for developing the proposed model key management infrastructure, by utilized the symmetric key management libraries, as compared with the OpenStack Swift this features not included. Fifth feature database, the database in the OpenStack Swift not secure but on the proposed model the database repository server is secure and is responsible for encrypting the saving and retrieving data. Sixth feature fault tolerance, these features is supported in both models. Seventh feature cross-platform these features is addressed in both models.

Table 2. Comparison of the characteristics proposed model and the OpenStack swift model

Features	Proposed model	OpenStack swift
Authentication enabled (symmetric encryption, hash function and CAPTCHA functionality)	Supported	Not Supported
Authorisation user-level	Supported	Supported
Keystone manager	Supported	Not Supported
PKI	Supported	Limited
Encrypted database	Supported	Not Supported
Fault tolerance	Supported	Supported
Cross-platform system	Supported	Supported

Conclusion

This paper discusses issues that arise with the OpenStack Swift; Swift enables users to manage their own data in a secure and centralised manner, which significantly facilitates the storage and access of personal data. The emergence of cloud computing has prompted Swift service providers to shift their user's applications and storage into the cloud to benefit from elastic resources and reduce operational costs. By storing users in the cloud computing platform, users lose physical control of their personal data, which requires each user to encrypt her/his data prior to uploading data to the cloud servers. The use of cryptographic and hash function algorithms for data security will enable secure and efficient access control to user data. It also enables the separation of security from the cloud infrastructure without losing the advantages of cloud computing. Proposed model gain the ability to store data in multi-tenant models and services.

Users must be identified by keystone before they are allowed to use any of the OpenStack storage services. This step guarantees a unique point of entry. Keystone decrypts usernames and passwords and provides each user with a unique token that enables access to the services for which they are authorised. At the second level (proxy), asymmetric encryption RSA is implemented and the third level DB employs a unique cryptographic algorithm to protect sensitive data storage on Swift. Cryptographic keys are sensitive data that are required on the cloud platform in different cases.

Funding Information

The authors have no support or funding to report.

Author's Contributions

Hala Albaroodi: All the work belongs to me is my PhD work.

Selvakumar Manickam: My supervisor he support me a lot.

Mohammed Anbar: He is post-doctorate, he advise me a lot.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Adam.Younglogic.Com, 2012. Keystone-should-move-to-apache. World Wide Web.

Albaroodi, H.A., S. Manickam and M.F. Aboalmaaly, 2013. The classification and arts of open source cloud computing: A review. Adv. Inform. Sci. Service Sci., 5: 16-25.

Armbrust, M., A. Fox, R. Griffith, A.D. Joseph and R. Katz *et al*., 2010. A view of cloud computing. Commun. ACM, 53: 50-58, DOI: 10.1145/1721654.1721672

AS, 2010. The Auth System.

Albaroodi, H., S. Manickam and P. Singh, 2014a. Critical review of OpenStack security: Issues and weaknesses. J. Comput. Sci., 10: 23-33. DOI: 10.3844/jcssp.2014.23.33

Albaroodi, H., S. Manickam and M.F. Aboalmaaly, 2014b. Security model for the OpenStack object storage: A review. Proceedings of the 1st International Engineering Conference on Developments in Civil and Computer Engineering Applications, (IEC' 14), Erbil, Iraq

Baset, S.A., C. Tang, B.C. Tak and L. Wang, 2013. Dissecting open source cloud evolution: An OpenStack case study. Proceedings of the 5th USENIX Workshop on Hot Topics in Cloud Computing, Jun. 25-26, USENIX, San Jose, CA.

Cigoj, P. and T. Klobučar, 2012. Cloud security and OpenStack. Proceedings of the 1st International Conference on Cloud Assisted Services, Oct. 22-25, Bled, Slovenia, pp: 20-27.

Cooper, J.D., 2013. Analysis of security in cloud platforms using OpenStack as case study. PhD. Thesis, University of Agder.

Chou, T.S., 2013. Security threats on cloud computing vulnerabilities. Int. J. Comput. Sci. Inform. Technol., 5: 79-88. DOI: 10.5121/ijcsit.2013.5306

Dai Yuefa, W.B., G. Yaqiang, Z. Quan and T. Chaojing, 2009. Data security model for cloud computing. Proceedings of the International Workshop on Information Security and Application, Nov. 21-22, Qingdao, China, pp: 141-444.

Hala, A.A., S. Manickam, M.F. Aboalmaaly and H. Palakarnim, 2013. A pilot study on Open Source Cloud Computing (OSCC) awareness in the Universiti Sains Malaysia education sector. IJACT, 5: 1264-1273. DOI: 10.4156/ijact.vol5.issue9.149

Khan, R.H., J. Ylitalo and A.S. Ahmed, 2011. OpenID authentication as a service in OpenStack. Proceedings of the 7th International Conference on Information Assurance and Security, Dec. 5-8, IEEE Xplore Press, Melaka, pp: 372-377. DOI: 10.1109/ISIAS.2011.6122782

Gellman, R., 2012. Privacy in the clouds: Risks to privacy and confidentiality from cloud computing. Proceedings of the World Privacy Forum, (WPF' 12), CiteULike.

Gansen, Z., R. Chunming, L. Jin, Z. Feng and T. Yong, 2010. Trusted data sharing over untrusted cloud storage providers. Proceedings of the 2nd International Conference on Cloud Computing Technology and Science, Nov. 30-Dec. 3, IEEE Xplore Press, Indianapolis, pp: 97-103. DOI: 10.1109/CloudCom.2010.36

Kim, J.M., H.Y. Jeong, I. Cho, S.M. Kang and J.H. Park, 2013. A secure smart-work service model based OpenStack for cloud computing. Cluster Comput., 17: 691-702. DOI: 10.1007/s10586-013-0251-1

Okuhara, M., T. Shiozaki and T. Suzuki, 2010. Security architecture for cloud computing. FUJITSU Sci. Tech. J., 46: 397-402.

Ponnuramu, V. and L. Tamilselvan, 2012. Data integrity proof and secure computation in cloud computing. J. Comput. Sci., 8: 1987-1995. DOI: 10.3844/jcssp.2012.1987.1995

Popovic, K. and Z. Hocenski, 2010. Cloud computing security issues and challenges. Proceedings of the 33rd International Convention (MIPRO), May 24-28, IEEE Xplore Press, Opatija, Croatia, pp: 344-349.

Rodriguez-Martinez, M., J. Seguel and M. Greer, 2010. Open source cloud computing tools: A case study with a weather application. Proceedings of the 3rd International Conference on Cloud Computing, Jul. 5-10, IEEE Xplore Press, Miami, FL, pp: 443-449. DOI: 10.1109/CLOUD.2010.81

Venkatesa, K.V. and S. Palaniswami, 2012. A Dynamic Resource Allocation Method for Parallel Data Processing in Cloud Computing. J. Comput. Sci., 8: 780-788. DOI: 10.3844/jcssp.2012.780.788