Kerberos-Based Authentication for OpenStack Cloud Infrastructure as a Service

Sazzad Masud and Ram Krishnan*
University of Texas at San Antonio
Sazzad.Masud@gmail.com, Ram.Krishnan@utsa.edu

Abstract

Cloud computing is an emerging technology, which will be a ubiquitous service in the near future. Cloud has also converged many seemingly different components such as compute, storage, etc. into a unified infrastructure. OpenStack is one of the prominent cloud computing software in the cloud community. It is deployed as Infrastructure as a Service, which means it allows users to provision their own machines in cloud by using its components, like storage, computation, etc. In order to provide such services, OpenStack needs to authenticate its users. The component in OpenStack that performs this function is called Keystone. In Keystone, the current mechanism is to provide a token to the requesting user, which is then provided to various other services from which the users request specific services (e.g. compute, storage, etc.) In this paper, a standard Kerberos-based authentication system is investigated and developed for OpenStack. A key contribution of this investigation is to gain understanding of the feasibility of Kerberos in OpenStack for the purpose of authentication. A major benefit is that the authentication system in OpenStack can then be based on a well-known and well-studied standard. A prototype authentication system of a component of the proposed protocol is implemented. The demonstration and evaluation of this implementation are also discussed.

Keywords: OpenStack, Kerberos, IaaS, Authentication, Cloud.

1 Introduction

Cloud computing is a convergence of information technology (IT). Various aspects of IT such as computing, networking, storage, applications, etc. have converged into a unified infrastructure called cloud. This has major implications to many other fields such as communication, banking, commerce, etc. since IT has become the backbone for enabling businesses to deliver service to their customers. This article focuses on improving the technology of cloud computing toward this end.

OpenStack [5] is an open-source Infrastructure as a Service (IaaS) [4] cloud computing software, where users can provision virtual machines by using its components such as storage (called "swift"), compute (called "nova"), etc. OpenStack began its journey on 2010 as a joint project of Rackspace and NASA [3]. This article is based on OpenStack. OpenStack is implemented in the python programming language. Figure 1 gives a high-level overview of OpenStack.

OpenStack can be deployed in standard hardware and its resources like computation, networking and storage are shared in the cloud. These resources can be controlled using a OpenStack dashboard. Users can avail these resources by using a client program such as an Internet browser. OpenStack has a modular architecture. Its components are listed below:

1. A Compute service codenamed Nova: Nova is a critical component of OpenStack. It is a cloud computing compute fabric controller. It supports a variety of virtualization technologies. It uses libraries such as Eventlet, Kombu and SQLAlchemy.

IT CoNvergence PRActice (INPRA), volume: 3, number: 2 (June), pp. 1-24

*Corresponding author: Department of Electrical and Computer Engineering, The University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249, Email: Ram.Krishnan@utsa.edu

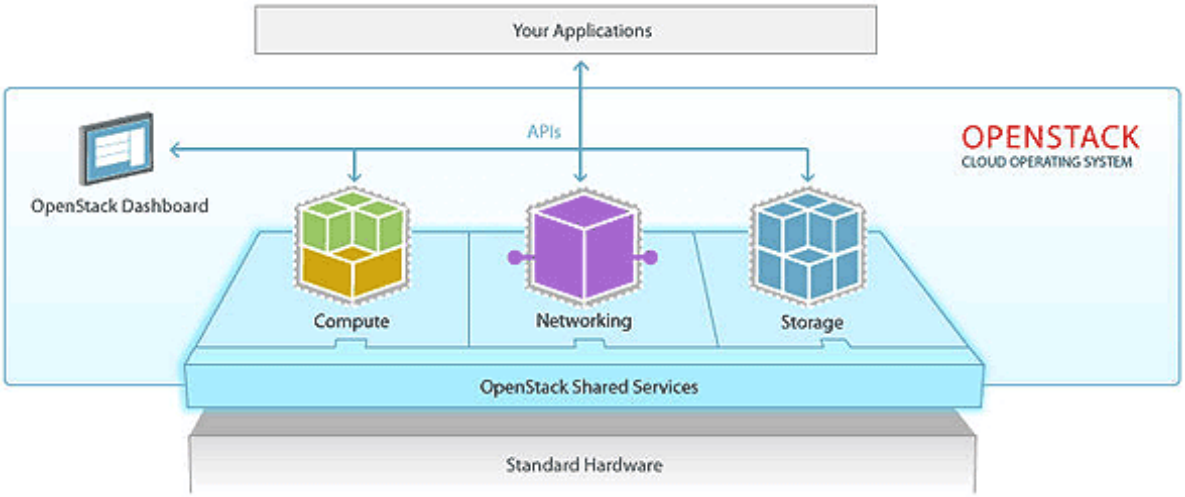

Figure 1: OpenStack Overview. Credit: `https://www.openstack.org/software/`

2. An Object Storage service codenamed Swift: Swift is a scalable storage system, and it is a highly available storage service. Organizations can store data in swift efficiently, safely and cheaply.

3. A Block Storage service codenamed Cinder: Cinder enables attaching local storage volumes for virtual machines provided by Nova.

4. A Networking service codenamed Neutron: Neutron manages virtual routers and allows users to create and manage their own virtual networks.

5. A Dashboard service for interfacing with OpenStack codenamed Horizon: Horizon is a web-based interface for OpenStack services, such as Nova, Swift, Glance, etc.

6. An Authentication and Authorization Service codenamed Keystone: Keystone is a shared service for authentication and authorization for the entire OpenStack cloud. Token creation, authentication, token validation, etc. are some of the major tasks performed and mediated by keystone.

7. An Image Service codenamed Glance: Glance allows creation and management of various operating system images using which new virtual machines cab be created by Nova.

A user can manage her OpenStack session using its dashboard (Horizon), which can be accessed using a web browser. To be able to get the services from various services, the user has to authenticate herself to the Keystone server.

It works as follows. First, the user gives her identity and credential (e.g. password) to Keystone. Assuming the user is registered, Keystone authenticates the user, creates a tamper-evident digital token that contains information about the user, the endpoint information of each service (e.g. Nova, Neutron, etc.), and the operations the user is allowed to perform at each of those services.

Keystone authentication is performed by using public key cryptography. It uses a digital signature, and the usage of the digital signature in this system is unconventional. It is well-known that a major drawback of digital signature is that it takes a longer time to sign and decrypt the data. For this reason in the real-world, a digital signature is used for small sized data (typically hashed data). But the existing system of keystone signs large amounts of data, and this makes the keystone's existing system an non-standard and inefficient for high-volume deployments.

The key point is that a significantly more efficient and standards-based authentication protocol for OpenStack can be developed. In this article, we demonstrate that this is feasible by re-designing and re-implementing OpenStack's authentication protocol implemented in its Keystone component, by employing the approach of the Kerberos protocol [1].

The Kerberos authentication protocol was initially developed at MIT. It is mainly designed for high volume servers. In Kerberos, a trusted third party, named Key Distribution Center (KDC), is used. KDC is an authentication server. All users of a network are connected to KDC. The KDC generates keys for the users and performs the authentication process. OpenStack also has an authentication server, named Keystone, and all the other services interact with this service. The similarity of OpenStack and Kerberos architecture makes us believe that the Kerberos protocol can be very effective for OpenStack. Kerberos has two protocols. One protocol uses symmetric key algorithm, here the entire authentication is done by using symmetric key. The other protocol uses public key cryptography. In this protocol, mutual authentication between the user and KDC is performed by public key cryptography. In this article, we investigate both of these protocols and based on these protocols we develop two respective authentication systems for OpenStack.

In this article, we also develop a prototype of the part of the Kerberos-based authentication system proposed in this article. Currently, Keystone uses digital signature on the entire data for authentication. This can be inefficient for large-scale deployments where high traffic is expected [3]. We propose the conventional approach of hashing the data before applying a digital signature. A hash algorithm converts a large amount of data into a small string of fixed size. Then this string is used for applying a digital signature. Thus, in our system Keystone signs a much smaller string instead of signing large data like the existing system.

We also conduct an implementation and evaluation of our designed system in OpenStack. We compare both the existing system and the newly implemented system with respect to time efficiency. We find that the token generation time and token validation time of our system is much more efficient that the current implementation in Keystone. Thus, the contributions of this article are as follows.

1. We investigate a Kerberos-based authentication system for OpenStack. Kerberos protocols are developed by using both symmetric key and public key cryptography. We investigate both protocols for OpenStack. Based on this investigation, we develop concrete and standard Kerberos-based authentication protocols for OpenStack.

2. We also develop a prototype of a small component of the Kerberos-based protocol for OpenStack.

3. Finally, evaluation and demonstration of the practicality and efficiency of the implemented parts of the protocols are demonstrated.

The reminder of this article is organized as follows. In section 2, we discuss the existing authentication mechanism in OpenStack in further detail. In section 3, we discuss the proposed Kerberos-based authentictaion mechanism. We discuss both a symmetric-key based and a public-key based mechanism. In section 4, we discuss our prototype implementation of specific component of the proposed mechanism in OpenStack. We conlcude in section 5.

2 Existing Authentication System in OpenStack

Keystone is one of the OpenStack components and this is used for identification, authentication and authorization service. This service is categorized into two primary functions.

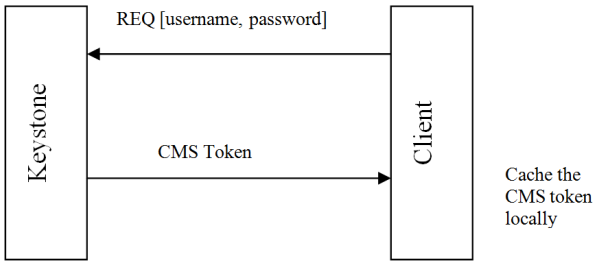

Figure 2: Logging into Keystone

1. **User Management:** keeps track of user's necessary data such as what roles the user has, which tenant the user belongs to, etc.

2. **Service Catalog:** keeps track of what services are available, and provides the location of the services' endpoints.

Keystone is structured as a group of internal services [2, 6]. These services are exposed on one or many endpoints. Keystone provides Identity, token, catalog, and policy services. A public key based mechanism is used in keystone's authentication system. Public key cryptography allows users to communicate securely over public networks and verify the identity of a user using digital signature. Digital signature is an electronic signature that can authenticate the user. In digital signature, a sender typically uses her private key to sign the data and the receiver uses the sender's public to key to verify the signature. A Certificate Authority (CA) plays the trusted role to vouch for the identity of the user with a specific public key.

In OpenStack, keystone can play the role of a CA using the keystone-manage utility or it can be done by a third party. A Keystone *PKI token* is used for authentication. A PKI token is nothing but a token signed by keystone with its signing or private key. Keystone uses cryptographic message syntax (CMS) with PKI. For this reason, the token is often referred to as CMS token. Whenever user authenticates with his/her user name and password, keystone gathers credential data (e.g. user's roles) of the user and creates a token and places them in a file called user metadata (see figures 18, 19, 20 21 22 23 in appendix). The metadata contains all information of the user like token, service catalog, user role, etc. It also contains an issue and expiration date and the id of the token. The tenant information follows next after which the service catalog information is placed. The service catalog has the information on the service(s) and their endpoints the authenticating user can avail. The endpoints are where the services should connect to obtain a specific service (e.g. compute vs network service). After the endpoints, the information about the user is listed. It shows the roles of the user, username and id of the user. Again, this data is called CMS data because the id of the services here is written in CMS format, and the signed CMS data is called CMS token.

When a user logs-in (figure 2) to her machine with her username and password, keystone gathers all of the above-mentioned information and generates a CMS token and sends it to the user's workstation. The user's OpenStack client program in her workstation caches the token locally and uses it for later requests. When the user later requests for a service using her client in the workstation (figure 3), the client sends the token along with the service request. The OpenStack service verifies the user's signature and responds back with the token.

When client needs any of the services like nova, glance, cinder etc., it sends a request along with the CMS token. The target service receives the CMS token and verifies the signature, and provides the requested service if the token is valid and the user is authorized.

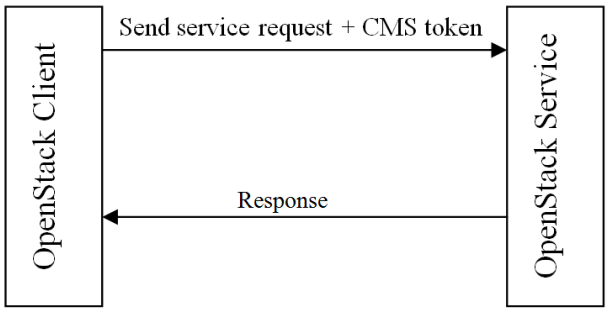

Figure 3: Requests for Service

Token Verification and Expiration PKI token enables services for offline verification. Three things are necessary to verify (refer figure 4):

Verify token signature: To verify the signature, OpenStack services endpoints need keystone's certificate. It can be obtained directly from keystone or by any third party if it is used in OpenStack. If services could not find the certificates locally, it can download it from keystone. With this certificate, it verifies the CMS token, and if it is a valid signature, it decrypts the metadata and checks the expiration date. If the signature is not valid, it rejects the request by sending an error (e.g. 401 HTTP).

Token Expiration Date: The OpenStack services analyzes the extracted metadata for validity. For example, it finds the expiration date and compares it against the current time.

Handling Revoked Tokens: Revoked tokens are enforced using a token revocation list (see figure 24 in appendix) in the keystone. OpenStack services update the revocation list frequently. The list format is in standard JSON. This list has an expiration date and time. The id field is the MD5 hash calculated on CMS user token that is revoked. Service endpoints calculate the MD5 hash of the presented user token and checks if it is present in the revocation list. If not, the token is considered as valid. After verifying the signature, token service endpoints reply to the client in according to the request made by the client. For instance, if the service endpoint is Nova and the requested service is to "Create VM", then Nova creates a VM on behalf of the user and responds with the appropriate information (e.g. IP address of the created VM).

Although the current approach of using digital signature on the whole token works for integrity and authentication, it also has some significant drawbacks:

1. The process of generation and verification of digital signature on the whole token takes a considerable amount of time. This can negatively impact the speed of communication of the receiver and sender.

2. Although the digital signature provides the integrity of the token contents, it cannot provide the secrecy of the data. We emphasize that tokens can contain sensitive information, including privacy sensitive information, and hence must be encrypted to preserve confidentiality.

3. Since a token cannot be reused, token generation is also significantly impacted due to the inefficient way digital signature in employed in the current implementation.

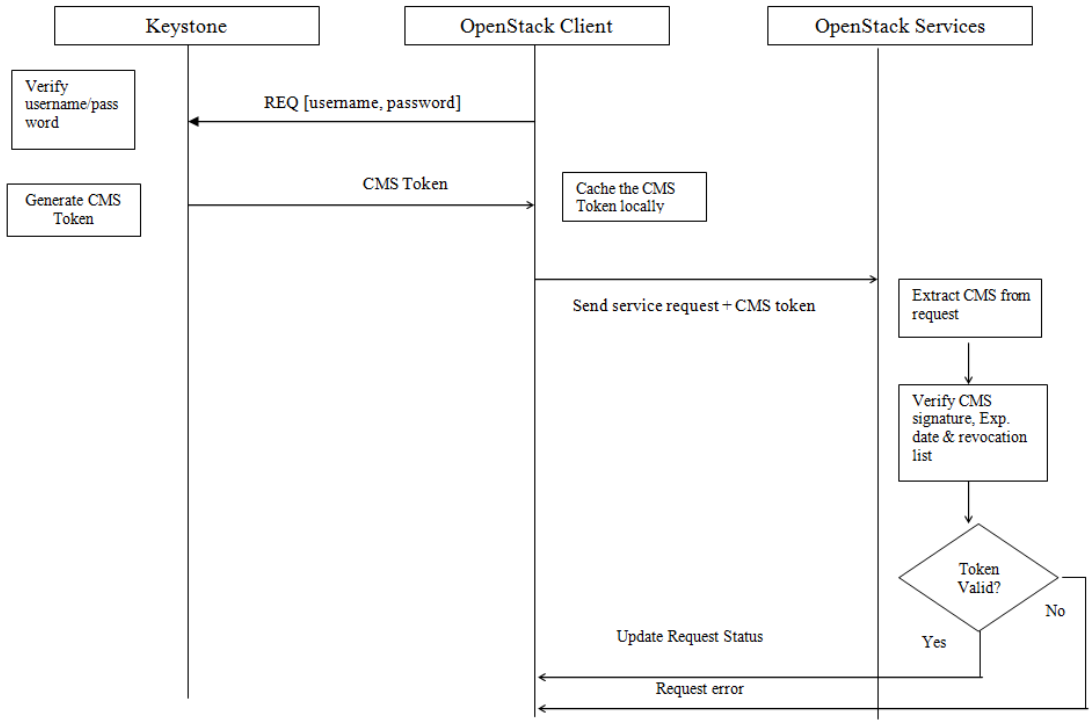

Figure 4: Process of Authentication for Keystone

3 Proposed Kerberos-Based Authentication System

The Kerberos protocol uses a central KDC (key distribution center) which acts as a trusted third party. In Kerberos, the KDC and the other entities use a "secure" clock for the purpose of detecting replay attacks and checking token validity. Kerberos uses timestamp as an authenticator. The clocks are assumed to be synced with a small amount of known clock drift. OpenStack has various components which are connected to each other, and there is also an authentication server, keystone, which provides authentication service to each component. Given a distributed set of services, we believe Kereberos is an appropriate architecture for enabling inter-service and user to service authentication. Kerberos can be configured to work with either using a symmetric key or a public key cryptosystem. In the following, we assume that the reader is familiar with the Kerberos protocol.

3.1 Kerberos Protocol with Symmetric Key

In this protocol (see figures 5, 6, 7, 8), a symmetric key cryptosystem is used and the KDC shares a secret key with each user and with each OpenStack service. The user logs-in to her workstation with her username and password. The workstation derives the key KA from the hash of the password and uses KA to get the Ticket Granting Ticket (TGT). The workstation sends a request that the user needs a TGT. The KDC then creates a session key and the TGT. Then it encrypts the session key and TGT with KA and sends them to the user's workstation:

1. A session key, S_A (secret key to be used in the login session).
2. A ticket granting ticket.

TGT contains the session key, the user's id, and expiration time, which are encrypted with K_{KDC}. So this ticket cannot be decrypted with anyone but KDC.

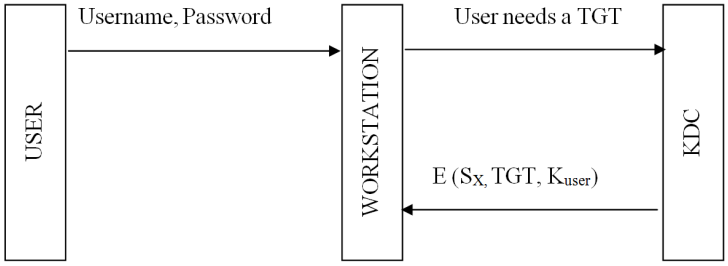

Figure 5: Login Process Kerberos

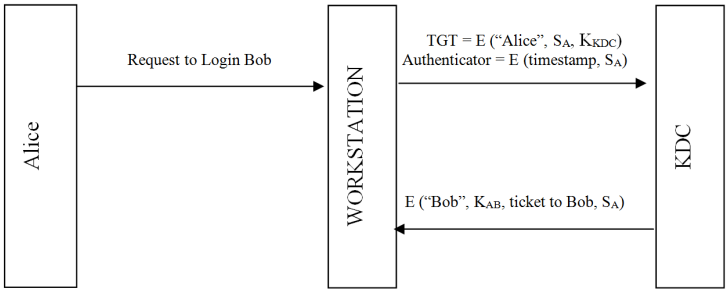

Figure 6: Getting ticket to Bob for Alice (Symmetric Key)

- TGT = E ("Alice", S_A, K_{KDC}).

Now if the user needs authenticate with any othe user she needs to gather another ticket from KDC. This will allow the user to communicate with other services. Let's assume Alice wants to communicate with Bob. So Alice's workstation sends KDC a request that says Alice needs to communicate with Bob. With the request, the workstation sends the TGT, the name Bob and an authenticator. Authenticator is a timestamp encrypted with the session key of Alice. That is, Authenticator = E (timestamp, S_A). KDC receives these data and generates K_{AB} (session key for Alice and Bob), and decrypts the TGT to get S_A (session key). The KDC then verifies the authenticator by using S_A. After verifying the timestamp, the KDC generates a ticket for Alice to communicate with Bob. This ticket is called ticket to Bob. This ticket contains Alice's name and K_{AB}. Both are encrypted with Bob's key, K_B (shared key of Bob).

- Ticket to Bob = E ("Alice", K_{AB}, K_B).

Then the KDC sends a message that includes Bob's name, K_{AB} and ticket to Bob. This message is encrypted with Alice's session key, S_A. When Alice's workstation gets the ticket to Bob, it sends a request to Bob. The request contains the ticket to Bob and an authenticator. This authenticator is timestamp encrypted with K_{AB}. Bob decrypts the ticket and gets the K_{AB}. Then Bob verifies the authenticator with K_{AB} and returns timestamp+1 encrypted with K_{AB}.

Kerberos in OpenStack: Similar to our previous discussion, when a client logs-in to her workstation with username and password, the workstation asks for a TGT to KDC. User's workstation will derive the shared key from the password of the user. KDC will generate a session key for a client, SC, and generate a TGT. The TGT contains username of the user and the session key, and both of them will be encrypted with keystones secret key. So TGT = ("client", S_C). Keystone will encrypt this data with user's shared key and send it to the client.

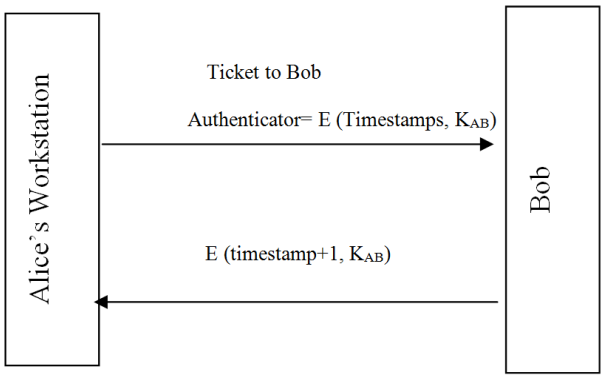

Figure 7: Logging into Bob's from Alice's workstation (Symmetric Key)

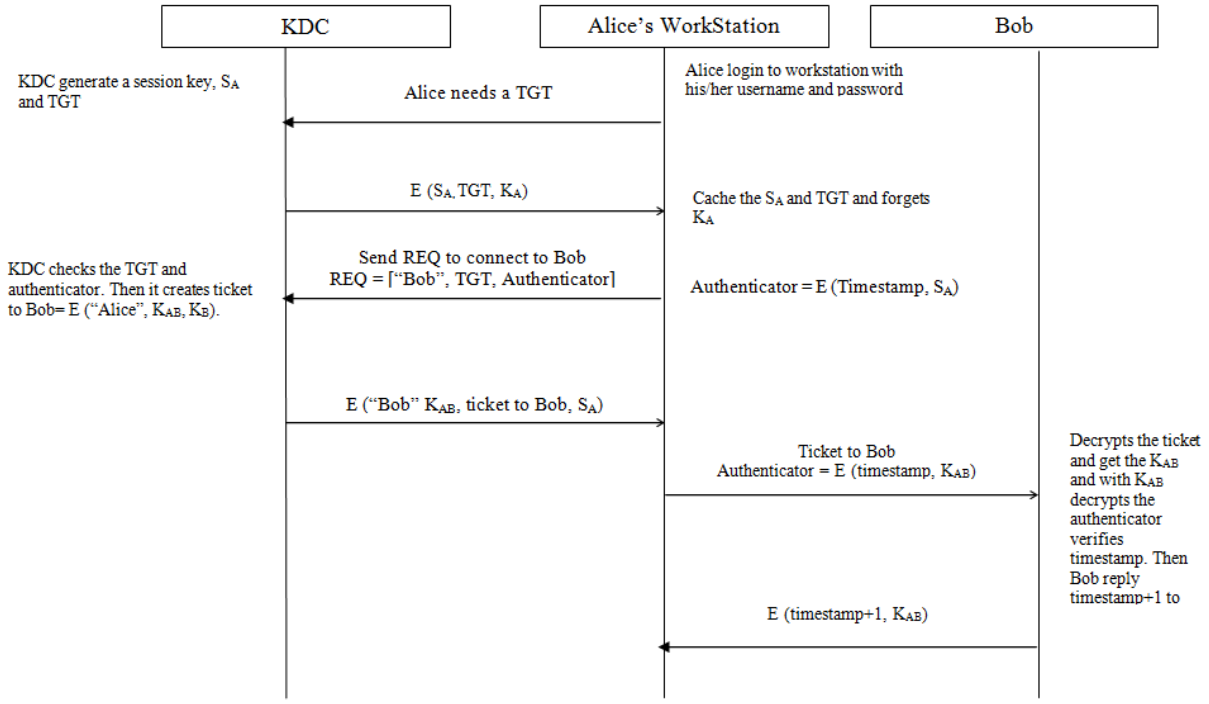

Figure 8: Processes of Kerberos with Symmetric Key Authentication

To connect to OpenStack services (figure 9), a client will send a request to keystone that has the service's name (e.g. Nova), TGT and an Authenticator. Authenticator contains timestamps encrypted with client's session key, S_C. So, Authenticator = E (timestamp, S_C). Keystone will receive the request and decrypt the authenticator and check the timestamp against the current time. KDC will then generate a key for both the OpenStack client and OpenStack services, K_{CN}. It will also generate a ticket for a client to connect to OpenStack service. This ticket will contain the hash of user metadata and the username of the client. The data, hash of the metadata and username will be encrypted with the service's shared key.

Ticket to service = E ("client", K_{CN}, h (user metadata), $K_{SERVICE}$).

Keystone will now send a message to the client that contains the service's name, key for client and

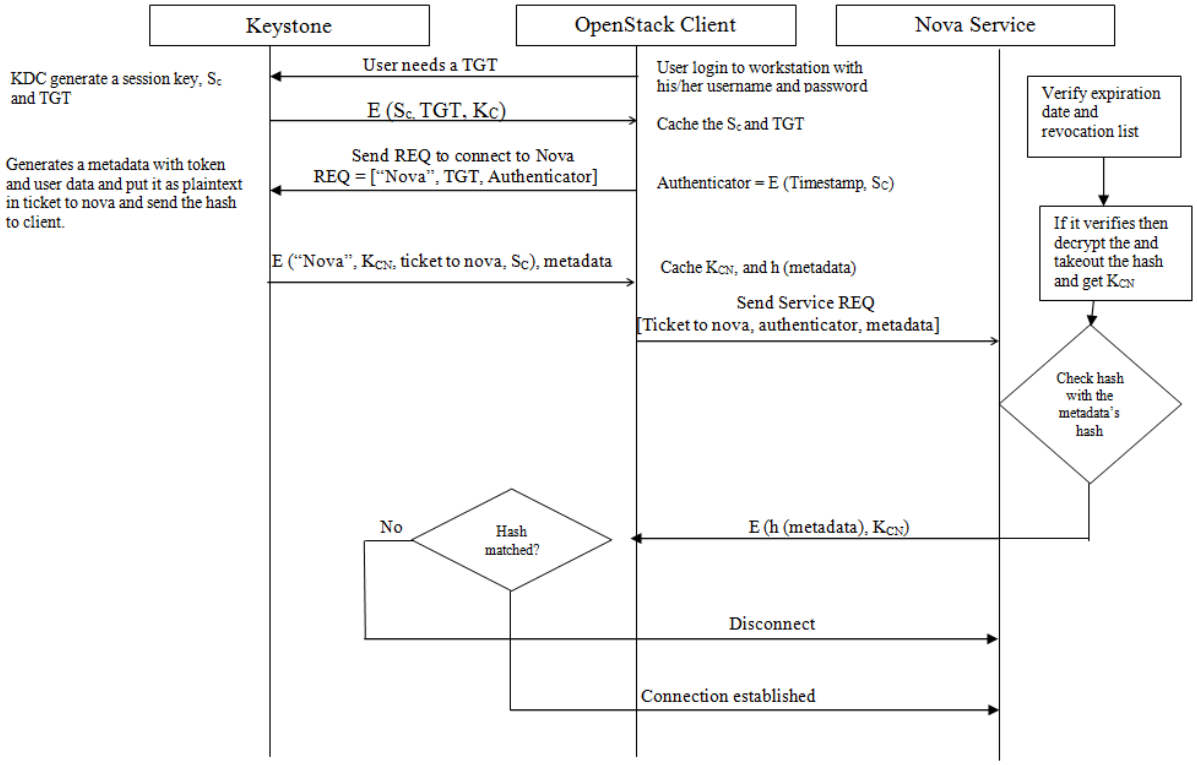

Figure 9: Kerberos-Based Authentication System with Symmetric Key for OpenStack

service K_{CN}, plaintext of user metadata. The session key between client and service, client's username and ticket to nova will be encrypted with S_C. User metadata will be sent in plain text.

To connect to the OpenStack services, the client will send a request to the service. A request will contain the ticket, client's username, an authenticator and the metadata. This authenticator is also a timestamp encrypted with K_{CN}. The OpenStack service will receive the request, check the revocation list and expiration date of the user metadata. Then it will decrypt the data and retrieve the hash and compare it with the known hash of the received plaintext data. A matche indicates that the request is valid. Then the OpenStack service will encrypt the hash data with K_{CN} and send it to the Client. With the hash data, the client will authenticate the service, and a connection will be established. This connection will be valid until the client logs off.

While the connection is established, the client does not need to send the token every time a service is needed. The session remains valid as long as the token remains valid.

3.2 Kerberos Protocol with Public Key

In this section we discuss the public key based protocol for Kerberos (see figures 11, 12, 13,). In public key infrastructure, every entity needs a public key and a private key. A certificate authority issues a certificate for every entity. In Kerberos, the KDC can act as a certificate authority. The KDC has all the certificates and public key of every entity, and all the entities have the certificate of KDC, so that they can verify the signature of KDC. KDC also has a secret key. Only KDC knows about this key, and no other entity has the knowledge of it. Suppose user Alice wants to talk to another user Bob. For communicating

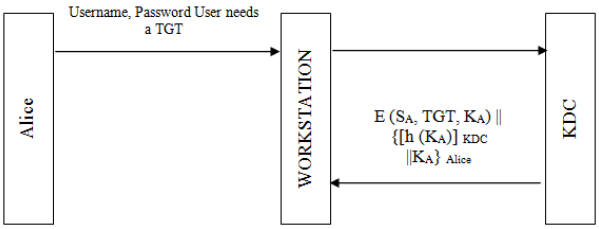

Figure 10: Login Process (Kerberos with Public Key)

with Bob, Alice has to log initially into her workstation. Her workstation will ask for TGT to KDC. This process is the same as Kerberos with symmetric key. In reply, KDC will send the TGT and session key. This message sending process is different from the previous Kerberos protocol. Here the TGT and a session key are encrypted with a key, which is generated only for Alice. KDC takes hash of this key, and signs it. Then it encrypts the hash and the plaintext of the key with Alice's public key and sends it to Alice. Then Alice's workstation receives the message. It decrypts the hash and the key. Then it takes the hash of the key and checks it with the received hash. If both the hashes match with each other, then it uses the key to decrypt the TGT and the session key. Alice's workstation then caches the TGT and session key locally.

- TGT = E ("Alice", SA, KKDC); SA is session key, and KKDC is a secret key of KDC.
- KA = key for Alice generated by KDC.

When Alice wants to communicate with Bob, Alice's workstation sends a request that she wants to talk to Bob with TGT and an authenticator to KDC. KDC receives the request and checks the TGT and verifies the authenticator. KDC issues a ticket to Bob and sends it to Alice. Ticket to Bob contains a key, and KB generated by KDC. KDC takes the hash of KB and signs it. Then encrypts the hash and the plaintext of KB with Bob's public key and places it in the ticket. The ticket also contains Alice's name, K_{AB} (the session key for Alice and Bob). Alice's name and K_{AB} are encrypted with K_B.

Ticket to Bob = $[h(K_B)]_{KDC}||K_B$ || Bob || E ("Alice", K_{AB}, K_B)

KDC encrypts the ticket to Bob, Bob's username and K_{AB} with the Alice's session key S_A and sends it to Alice. Alice's workstation decrypts the message and caches the ticket to Bob and K_{AB} locally.

Alice sends the ticket to Bob and an authenticator which is encrypted with K_{AB}. Bob receives the ticket, verifies the signature of KDC and retrieves K_B. Bob then decrypts K_{AB} with K_B and stores it locally. Finally, Bob verifies the authenticator and sends another authenticator to Alice as a response to her challenge. Alice verifies the authenticator and establishes a connection.

Kerberos with Public Key for OpenStack: Kerberos with public key cryptography can be used in OpenStack (see figures 14). Here Keystone can act as KDC and also as a Certificate Authority. Keystone will have all the entities' public keys and the certificate. Similarly, all the entities have Keystone's public key. When user logs into his workstation, the workstation sends a request to keystone that user needs a TGT. Keystone generates a session key for the user and sends the TGT and the key to client.

- TGT = E ("User", S_U, $K_{keystone}$)

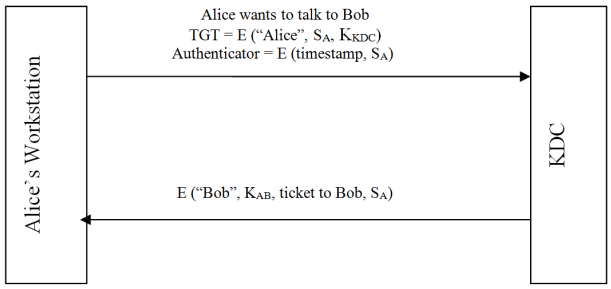

Figure 11: Getting Ticket to connect Bob

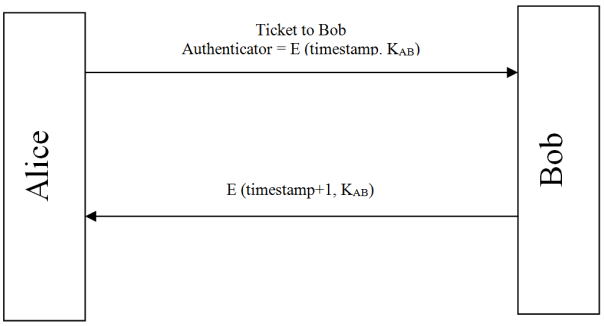

Figure 12: Sending Ticket to Bob

- $K_{keystone}$ = secret key for Keystone
- S_U = session key for user and keystone.

The OpenStack client receives the message and decrypts KU and retrieves the hash. It checks the hash with the hash of the KU. Then it uses KU to decrypt TGT and SU and stores these locally. When a service is needed from, say Nova, the user will send the TGT, along with a request to keystone. Keystone will check the TGT and grant a ticket for Nova. Then Keystone will send a ticket to the user.

- Ticket to Nova = $\{[h(K_{Nova})]_{Keystone}||K_{Nova}\}_{Nova}$ —— E {"User", User metadata, K_{UN}, $K_{Nova}\}$

The OpenStack client sends the ticket to Nova and an authenticator to Nova. Nova verifies the signature of keystone and finds K_{Nova} and the hash. It checks the hash with the hash of K_{Nova}. If both match, Nova uses K_{Nova} to decrypt user metadata and session key K_{UN}.

4 Prototype Implementation in OpenStack and Evaluation

A major issue with the implementation of the digital signature in the current implementation of Keystone is that the signature is applied on the entire token. A better approach is to hash the token and apply the signature on the hash. For the implementation of the keystone authentication system, we take the hash of the metadata, which is created by keystone. Keystone uses its private/signing key to sign this data after taking the hash of the metadata.

The format of the data is the same as the CMS token format. Instead of sending this CMS token as in the original system, this system also sends the original data with it. The id of the token is "placeholder"(see figure 25 in appendix). It is being replaced by the hash of the metadata and sent to the client

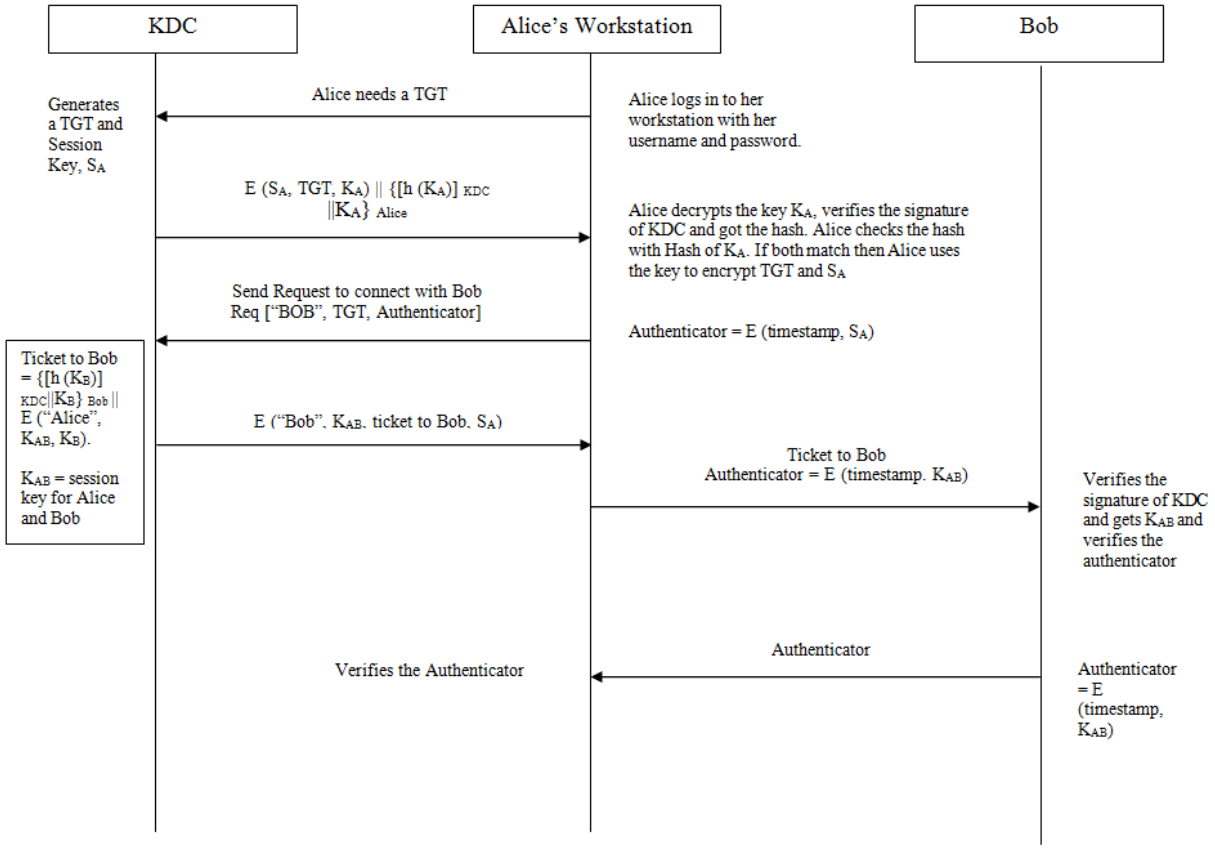

Figure 13: Kerberos with Public Key

(see figure 26 in appendix). The client receives this data and caches it locally. When the client requests a service, it sends a request with this data. The Nova service then verifies the data and replies to the client with the requested service.

Verification of the signature on the Hashed Token: First, Nova receives the metadata and extracts the hashed data from the id (see figure 15). Then it verifies the validity of the keystone's signing certificate, which is stored in Nova service endpoints. Nova gets the hash of the original metadata after it verifies the signature. Then it changes the id of the token of the metadata to "placeholder" and hashes the data again. If both hashes match, it proceeds to the next step and if it does not match it returns an error.

Token Expiration Date: If the signature check succeeds, the Nova service checks the expiration date of the token. That is, it finds the expiration date and compares it against the current time.

Handling the Revoked Token: This part is done exactly the same as previously explained. It takes the hash of the user token and checks it with the token in the revocation list. If the hash does not match with the revocation list, the token is valid and the service replies to the request.

Performance: We measured the performance difference between the original implementation and a new implementation with the proposed changes where the digital signature is applied to the hash of

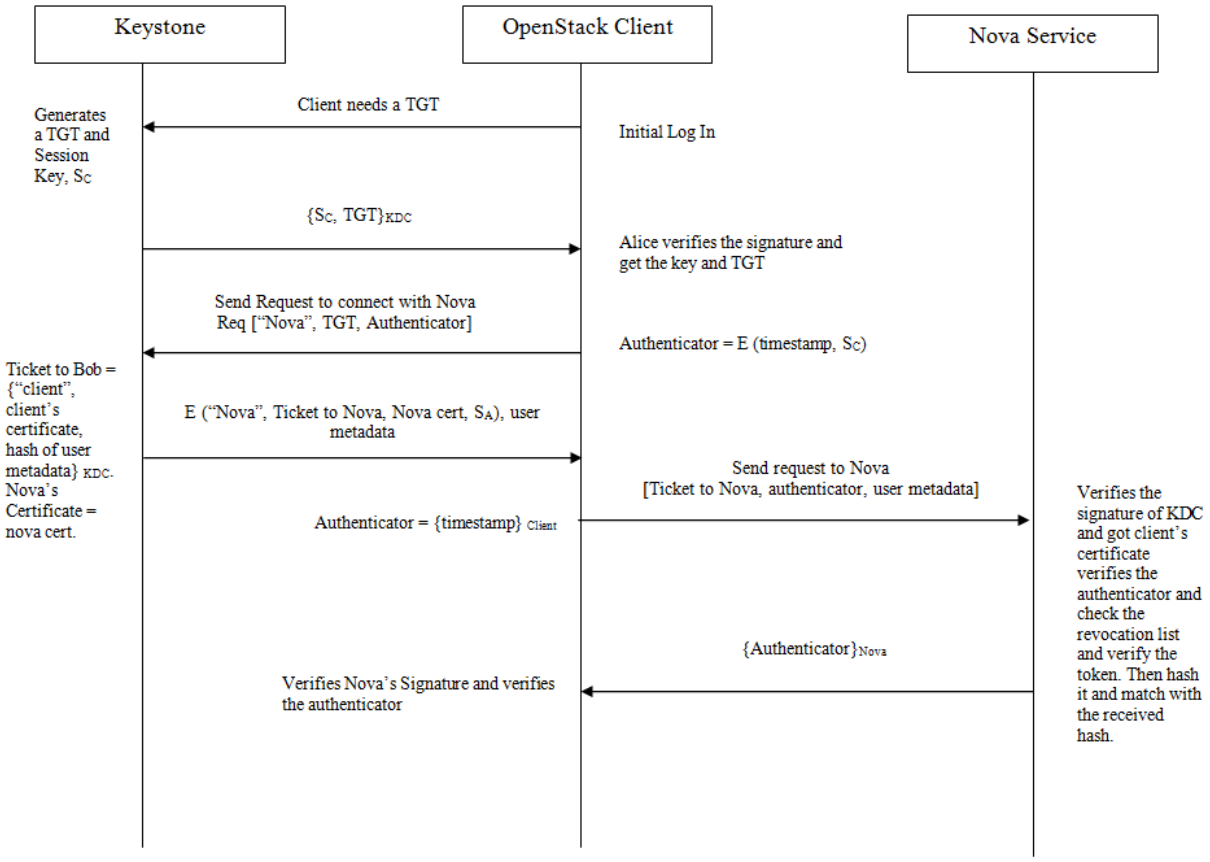

Figure 14: Kerberos with PKI in OpenStack

the token, instead of the token on the whole. A timer was placed in both the systems, to evaluate the performance. The timer calculates the token generation time and token validation time. For token generation, the implemented system is relatively efficient, which is a positive result. This is because, token generation in the implemented system requires the additional work of hashing but yet required almost the same amount of time. The graph in figure 16 compares the token generation time for the existing model and implemented model over hundreds of requests.

As mentioned above, in the existing model, user metadata is directly signed. In the new model, user metadata is hashed first and then signed. This is the reason for taking more time in the model. The difference between these two models in token generation is very negligible. But for validating the token the existing model takes significantly more time than the implemented model. Hash is a smaller string than the original metadata. So decrypting the hash data takes less time. Furthermore, verifying the digital signature on the hashed data also takes significantly lesser amount of time. Thus, in the newly implemented model, OpenStack services such as Nova, are capable of handling requests much more efficiently than the existing OpenStack implementation. The graph in figure 17 shows the difference between the token validation time of the two systems.

From the graph, we can see that the existing system takes almost 10ms longer to validate a token each time. While 10ms for one request is negligible but when we are sending 500 requests or more, the exiting system is losing a lot more time than the implemented system. As mentioned earlier, the reason for these differences is the unconventional use of digital signature in Keystone. Digital signature is typically applied on a small amount of data like hash. But in keystone a huge blob of data is signed,

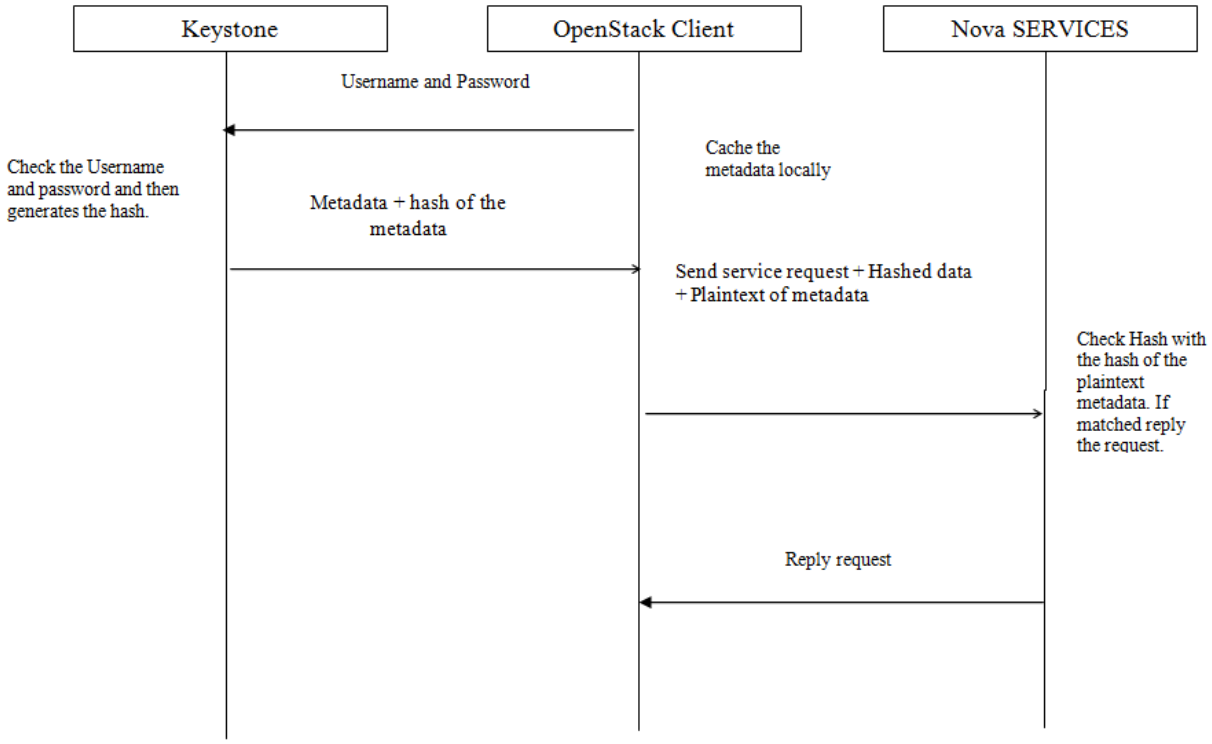

Figure 15: Implemented Authentication System for OpenStack

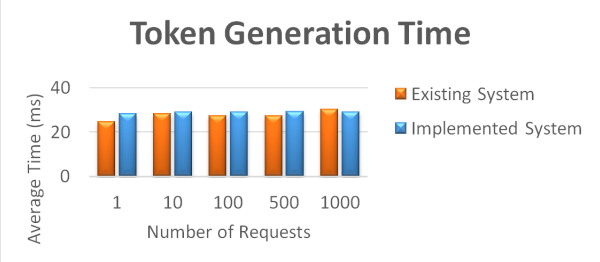

Figure 16: Token Generation Time

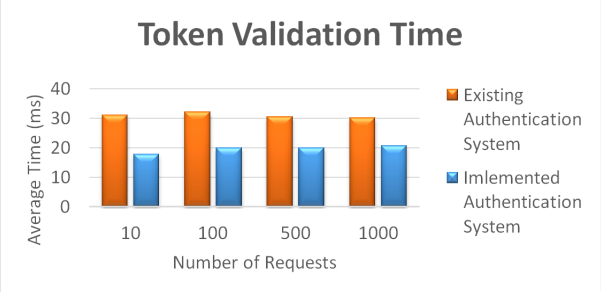

Figure 17: Token Validation Time

which takes much a longer time. This is one of the major drawbacks of digital signature. But, on the other hand, hash is a very small piece of data. This allows for efficient creation and verification of signature.

5 Conclusion and Future Work

OpenStack is a prominent open-source software for infrastructure as a service cloud. The usage of this software is increasing day by day. So it is necessary to investigate the security architecture. We find that the existing authentication system in OpenStack is unconventional, and can take a significant amount of time to authenticate that what is ideal. Our hypothesis in this article was that the existing system is time inefficient, and a time efficient authentication protocol can be developed for OpenStack using well-known authentication protocols such as Kerberos. The data from our implementation of both systems shows that the existing system is not a time efficient system, and a time efficient system is developed for

OpenStack. This new system is a time efficient system without hampering any security issues.
OpenStack can be deployed in a high volume server in real-world scenarios where its services may process a large number of requests from multiple users from a single and across multiple tenants. Hence, it will need a more efficient and fast authentication system. Kerberos can be a good replacement for the current protocol of OpenStack. Kerberos is designed for high volume servers and Kerberos also provides mutual authentication between the sender and receiver in distributed settings. Designing the OpenStack authentication system with Kerberos will make OpenStack more secure and can improve the performance of its services.

In the future, we plan to build on our current implementation. We plan to fully implement and evaluate the proposed Keberos protocol in Openstack, where each of its services can act as a user in Kerberos, and Keystone as the KDC with which each service has a registered key.

Acknowledgments

We are very thankful to Mr. Farhan Patwa, Associate Director of the Institute for Cyber Security (ICS) at the University of Texas at San Antonio, for providing major support toward the implementation of our proposed system in an OpenStack-based cloud at ICS.

References

[1] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World*. Prentice Hall, 2002.

[2] B. Kupidura. Understanding OpenStack Authentication: Keystone PKI. `https://www.mirantis.com/blog/understanding-OpenStack-authentication-keystone-pki/`, July 2013.

[3] S. Mandy. Drawbacks of digital signature. `http://computerfun4u.blogspot.com/2009/02/drawbacks-of-using-digital-signature.html`.

[4] P. Mell and T. Grance. The NIST Definition of Cloud Computing. Technical Report 800-145, National Institute of Standards and Technology, 2011.

[5] Rackspace. OpenStack: The Open Source Cloud Operating System. `http://www.openstack.org/software/`.

[6] R. Xu. Keystone authentication. `http://OpenStackoz.blogspot.com/2014/08/keystone-authentication.html`.

Author Biography

Sazzad Masud is from Dhaka, Bangladesh. He received his MS in Electrical Engineering for UTSA in 2014.

Ram Krishnan is Assistant Professor of Electrical and Computer Engineering at the University of Texas at San Antonio, TX, USA. His research includes foundational aspects of computer security such as Access control, Security Models, Security Policy Enforcement, Formal Policy Analysis, etc. His current research is mostly focused on security aspects of Cloud Computing and mobile platforms.

Appendix

Figures 18, 19, 20, 21, 22, and 23 below show the metadata structure of a user in JSON format.


```
{
   "access":{
      "token":{
         "issued_at":"2014-10-28T21:23:21.890935",
         "expires":"2014-10-28T22:23:21Z",
         "id":"placeholder",
         "tenant":{
            "description":null,
            "enabled":true,
            "id":"989a962f3b8d4b63be40f18df0ed8147",
            "name":"admin"
         }
      },
      "serviceCatalog":[
         {
            "endpoints":[
               {
                  "adminURL":"http://10.245.122.123:8774/v2/989a962f3b8d4b63be40f18df0ed8147",
                  "region":"RegionOne",
         }
      },
      "serviceCatalog":[
         {
            "endpoints":[
               {
                  "adminURL":"http://10.245.122.123:8774/v2/989a962f3b8d4b63be40f18df0ed8147",
                  "region":"RegionOne",
```


Figure 18: User Metadata

```
"internalURL":"http://10.245.122.123:8774/v2/989a962f3b8d4b63be40f18df0ed8147",
                  "id":"0993d102d0794d8bbee6079d63796297",
                  "publicURL":"http://10.245.122.123:8774/v2/989a962f3b8d4b63be40f18df0ed8147"
               }
            ],
            "endpoints_links":[            ],
            "type":"compute",
            "name":"nova"
         },
         {
            "endpoints":[
               {
                  "adminURL":"http://10.245.122.123:8776/v2/989a962f3b8d4b63be40f18df0ed8147",
\                 "region":"RegionOne",

"internalURL":"http://10.245.122.123:8776/v2/989a962f3b8d4b63be40f18df0ed8147",
                  "id":"156ab6e0f7d34e91b09022f153f9628e",
                  "publicURL":"http://10.245.122.123:8776/v2/989a962f3b8d4b63be40f18df0ed8147"
               }
            ],
            "endpoints_links":[


            ],
            "type":"volumev2",
            "name":"cinderv2"
         }, internalURL":"http://10.245.122.123:8774/v3",
                           "id":"088a81c6dcc04d08a0034e420988451f",
                          "publicURL":"http://10.245.122.123:8774/v3"
}
           ],
```


Figure 19: User Metadata (continued)

```
            "endpoints_links":[

            ],
            "type":"computev3",
            "name":"novav3"
         },
         {
            "endpoints":[
               {
                  "adminURL":"http://10.245.122.123:3333",
                  "region":"RegionOne",
                  "internalURL":"http://10.245.122.123:3333",
                  "id":"0cef25ca075e404cb53bfaeb745e9bb3",
                  "publicURL":"http://10.245.122.123:3333"
               }
            ],
            "endpoints_links":[

            ],
            "type":"s3",
            "name":"s3"
         },
            "endpoints":[
               {
                  "adminURL":"http://10.245.122.123:9292",
                  "

      region":"RegionOne",
```


Figure 20: User Metadata (continued)

```
                  "internalURL":"http://10.245.122.123:9292",
                   "id":"07fe08574c564486a32d46d34a7de6b0",
                   "publicURL":"http://10.245.122.123:9292"
               }
            ],
            "endpoints_links":[

            ],
            "type":"image",
            "name":"glance"
         },
         {
            "endpoints":[
               {
                  "adminURL":"http://10.245.122.123:8776/v1/989a962f3b8d4b63be40f18df0ed8147",
                  "region":"RegionOne",
"internalURL":"http://10.245.122.123:8776/v1/989a962f3b8d4b63be40f18df0ed8147",
                  "id":"552c5ec98600485b9166a8a2a53b5fec",
                  "publicURL":"http://10.245.122.123:8776/v1/989a962f3b8d4b63be40f18df0ed8147"
               }"endpoints_links":[

            ],
            "type":"volume",
            "name":"cinder"
         },
         {
            "endpoints":[              {
                  "adminURL":"http://10.245.122.123:8773/services/Admin",
                  "region":"RegionOne",
                  "internalURL":"http://10.245.122.123:8773/services/Cloud",
```


Figure 21: User Metadata (continued)

```
                "id":"00135317c162467f8f8430fce071dbff",
                "publicURL":"http://10.245.122.123:8773/services/Cloud"
             }
          ],
          "endpoints_links":[

          ],
          "type":"ec2",
          "name":"ec2"
       },
       {
          "endpoints":[
             {
                "adminURL":"http://10.245.122.123:35357/v2.0",
                "region":"RegionOne",
                "internalURL":"http://10.245.122.123:5000/v2.0",
                "id":"00ffdb3c377e4865a5a1b1de1458edda",
                "publicURL":"http://10.245.122.123:5000/v2.0"
             }
          ],
          "endpoints_links":[
          ],
          "type":"identity",
          "name":"keystone"
       }
    ],
    "user":{
       "username":"admin",
       "roles_links":[],
       "id":"d07b7dfc1ecf45daba75b907447eb8d5",
       "roles":[
```


Figure 22: User Metadata (continued)

```
                {
                "name":"_member_"
            },
            {
                "name":"admin"
            }
        ],
        "name":"admin"
    },
    "metadata":{
        "is_admin":0,
        "roles":[
            "9fe2ff9ee4384b1894a90878d3e92bab",
            "14b7d2b9f2064ba9986e6749f6f34dfc"
```


Figure 23: User Metadata (continued)


```
{
   "revoked": [
       {
           "expires": "2014-11-1T08:31:37Z",
           "id": "aef56cc3d1c9192b0257fba1a420fc37"
       }
    …
   ]
}
```


Figure 24: Revocation List

```
{
   "access":{
      "token":{
         "issued_at":"2014-10-28T21:23:21.890935",
         "expires":"2014-10-28T22:23:21Z",
         "id":"placeholder",
         "tenant":{
            "description":null,
            "enabled":true,
            "id":"989a962f3b8d4b63be40f18df0ed8147",
            "name":"admin"
         }
      },
      ...
      ...
```


Figure 25: User Token

```
{
   "access":{
      "token":{
         "issued_at":"2014-10-28T21:23:21.890935",
         "expires":"2014-10-28T22:23:21Z",
           "id":"
MIIB2QYJKoZIhvcNAQcCoIIByjCCAcYCAQExCTAHBgUrDgMCGjAxBgkqhkiG9w0BBwGgJAQiIjA5M
DQ3YjBmZjI1ZmEzOWI5NzU0ODViOWQyYWEyZWNjIjGCAYEwggF9AgEBMFwwVzELMAkGA1UEBhMCVV
MxDjAMBgNVBAgMBVVuc2V0MQ4wDAYDVQQHDAVVbnNldDEOMAwGA1UECgwFVW5zZXQxGDAWBgNVBAM
MD3d3dy5leGFtcGxlLmNvbQIBATAHBgUrDgMCGjANBgkqhkiG9w0BAQEFAASCAQBLIgfVMj5NPboy
pnAY0mYo3NI8Bn9ys5v6okqEyCCbxIOFYVvYzjGWPmg3Qs-
X161WzPLDLrRnITLRZsjJcptozRZ4l8tgDO+z+zCXfPe6khjFf-
LAB3uQRfiKCQ7hqmogp5hLQatW561MciPXumwSHxQQIWtlmb-
Aek19NQ0mJhYLwNzirxsRf9v99GV0--JqCSZ6XQh-Fo66hyc8NOm9wxuNpz-
1WfWZommHjQRrzUuSeN67D6kWWz+deAAVIwQwm+n26Z3hTH3pVLGqB5I4LI1rbNbkwJw9bm+g4swQ
T2PYyDw+86WMtKJyKx4hEiFkDP8gSgfOZ1VT3WCGOFdN
",
         "tenant":{
            "description":null,
         ...
```


Figure 26: User Token (Implemented System)