Cryptographic Role-Based Access Control for Secure Cloud Data Storage

Aditya Kale
akale3@hawk.iit.edu

Ashish Deshpande
adeshp16@hawk.iit.edu

***Abstract* – Cloud data storage has been proven beneficial in storing big amount of data in a cost effective manner. While storing the data on public cloud protecting the privacy of data is a big challenge in today's world. This project mainly focusses on Role-Based Access Control Systems (RBAC) which ensures that the data stored on cloud will only be accessed by those users who has right access policies to retrieve that information. We have used role based encryption (RBE) scheme to implement this Role-Based Access Control model. RBE scheme is designed by using asymmetric bilinear groups. In this project we will see how a hybrid cloud structure can be used to enforce RBAC System. This new RBAC model allows user to store data securely on public cloud, while maintaining all the sensitive information on private cloud. We have demonstrated a symmetric key encryption using AES technique for encrypting and decrypting images. Our implementation shows that a role which is inherited from another can also access encrypted data of those roles in a Role hierarchy.**

***Keywords* – Security, Access Control, Role, Role Based Encryption (RBE), Role Based Access Control (RBAC), User Role Management.**

I. Introduction

Recently there has been a rapid growing trend of using online services. Major benefit of that is user can store the data online and can access his information from anywhere. However many online service provider does not have that large storing capacity to store data growing at such a fast rate. Data growth is the biggest data center hardware infrastructure challenge for large enterprises. Due to complexity and high maintenance cost, handling such a huge amount of data becomes more difficult. So most of the enterprises are moving towards the cloud storage services which provide solution to address this issue.

By outsourcing the user data to public cloud enterprises or online service provider can focuses more on the design of an application. Instead of worrying about the data management they can concentrate on different functionalities to improve user experience. Cloud services also provide the on demand resources for storing the data based on the need, so maintenance cost for handling such huge amount of data is also gets reduced.

However as public cloud is an open platform it leads to malicious attacks from both outsiders and insiders. Because of such attacks there arise several security issue such as how to control access of different outsider to a data stored on cloud. Also the owner of the data does not want to expose the original data and its content to the cloud. One way to provide solution to above problem is by giving a right access to a right user by implementing access control models.

Over the past few years there has been many access control models developed such as Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role-Based Access Control (RBAC). Many encryption scheme such as Role Key Hierarchy (RKH) and Role-Based Encryption (RBE) has been used to implement these access control models. Among all these, RBAC is proven as the best alternative to other traditional access control models.

II. Role Based Encryption

In role based access control model roles are mapped to access permissions and user are mapped to a particular role. Users are given roles based on their responsibilities in the organization. Permissions are given to roles instead of individual user. Users who has a specific role only access to the data. RBE scheme is used to develop a RBAC system. In RBE Roles are arranged in a hierarchy like a tree structure, where one role can inherits properties from other roles. RBE helps to enforce the RBAC policies on encrypted data stored on cloud. Since being first formalized in 1990's, RBAC has been widely used in many systems to provide users with flexible access control management, as it allows access control to be managed at a level that corresponds closely to the organization's policy and structure.

This RBE scheme is able to handle role hierarchies where one role can inherits the properties from another role. In these scheme user can be added at any time in a specific role even after the owner has encrypted the data. Owner does not have to re-encrypt the data if want to add new user to any role. Also a user can be revoked at any time from the role, then on onwards that user won't have any further access to the encrypted data. Removing the user does not affect another users and the role hierarchy.

A. Example of Role Based Encryption

The below figure shows the RBE example. There are 4 roles created in a hierarchical structure.

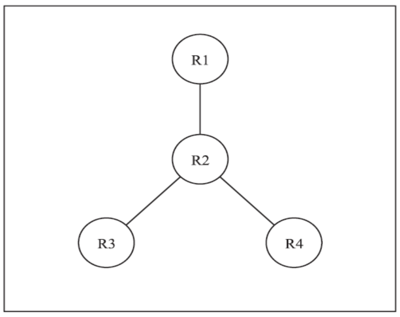

Figure 1 RBE Example

In above figure Role R2 inherits the properties from Role R3 and R4, and Role R1 inherits from Role R2. Let suppose Owner runs an encryption logic and encrypts the Message M using role R3 and outputs the cipher text tuple C in the cloud. Assume that role R1 has a set of user members {U1, U2, U3} and among them User U1 wants to decrypt the data encrypted by an owner. Now because role R1 is inherited from R2, and R2 is inherited from R3, user U1 from role R1 is allowed to access and decrypts the message M. U1 executes the decryption algorithm on message M by using its own decryption key and can successfully decrypt the message. In this scenario user do not require role parameters of the role by which the message was encrypted (i.e. Role parameters of R3). Just using the role parameters to which they belong (i.e. Role parameters of R1), user can decrypt the message. From above example we can see how RBE support the role inheritance in the data decryption.

B. Role Based Encryption System Architecture

In this section we will see the architecture of a secured cloud storage system. It's a hybrid cloud structure comprise of Public cloud and a Private cloud. Figure 2 shows the RBE system architecture.

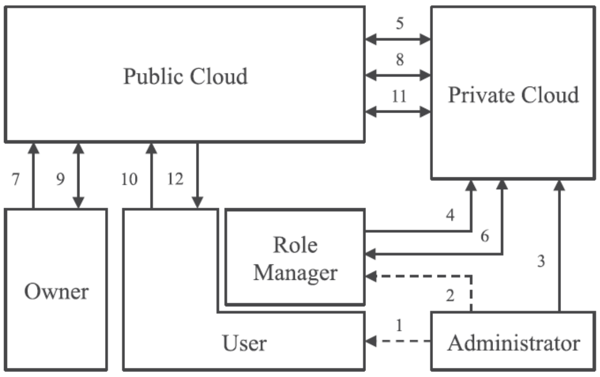

Figure 2 RBE System Architecture

In this architecture Private cloud stores all the sensitive data such as role hierarchy of the organization and also all the permissions given to a role. It also stores all the role user membership relations, where it is defined that which users has given which role access. On the other hand public cloud stores all the encrypted data and public parameters which are associated with the RBE scheme. The owner who wants to encrypt their data and user who wants to decrypt that data talks only with the public cloud. Thus it reduces the attack on private information stored on private cloud, providing another level of security.

This architecture comprise of six major components. Public Cloud, Private Cloud, System Administrator, Role Manager, Owner and a User. We can see that user and owner interacts only with the public cloud, where administrator and a role manager has only access to private cloud. We will first consider all the component specified in an architecture and will focus on their functionalities.

Public Cloud: Public cloud is a third party storage service provider which resides outside the organization's infrastructure. Online service provider outsource their encrypted data to the public cloud. As public cloud are open to all users their data can be accessed by any unauthorized user. All users which resides in the same public cloud and also by the employee of the cloud provider. Thus only the encrypted data and public information is stored on public cloud.

Private Cloud: Private cloud is built on an internal data center that is hosted and operated by a single organization. The organization only stores critical and confidential information in the private cloud [2]. As this cloud stores only role hierarchy and the role-user relations, amount of data stored on this cloud is relatively small as compared to public cloud. Also it requires very less computational power as it has to handle very less amount of data. Only Role manager and administrator has an access to the private cloud. No user or an owner gets a direct access to the private cloud which helps to reduce the attack on the organizations role hierarchy structure and user membership which are used in decision making.

System Administrator: The administrator is the certified authority of the organization [2]. Main role of a system administrator is to generate all necessary credentials, system parameters and store them on a cloud. System administrator handles the role hierarchy structure of an organization and store it in a private cloud. Whenever any role is changed in a hierarchy, administrator take care to update the role parameters in a private cloud.

Role Manager: Role manager is an entity who manages the relationship between the user and their roles. Also if any role memberships are changed role manager has to re compute the role parameters and update them in private cloud. Role manager performs add user and revoke user functionalities. These functionalities are used to add user in a role or to remove any user from a particular role. Adding or removing user operations does not affect any of the user. Thus there is no need to role manager to interact with any of the user.

Owner: Owner are the parties who wants to encrypt their data and store in the cloud so that other user can access that data. Owner sends a user information to a role manager specifying which users can access the data in terms of role based access policies. They are parties who specify relation between the permissions and roles.

User: User are the parties who wants to decrypt the data which is stored on the cloud by an owner. Before sending the encrypted data to a user, each user is first authenticated by a system administrator. After successful authentication administrator sends a secret key which is associated with a user identity. This secret key is used by user to form a decryption key which helps to decrypt the data from the cloud.

C. RBE Setup

For the implementation of this RBE scheme concept of bilinear pairings is used. Let G1, G2, GT be three cyclic groups of prime order p, and GT be a cyclic multiplicative group of prime order p. g and h are two random generators where g ∈ G1, h ∈ G2.
A bilinear pairing ê: G1 × G2 → GT satisfies the following properties:

- Bilinear: for a, b ∈ Z_P^* we have
  $\hat{e}(g^a, h^b) = \hat{e}(g, h)^{ab}$.
- Non-degenerate: ê (g, h) ≠ 1 unless g = 1 or h = 1.
- Computable: the pairing ê (g, h) is computable in polynomial time.

Two secret values such that s, k ← Z_P^* and two hash functions H1: $\{0, 1\}^* \rightarrow Z_P^*$, H2 : GT → G1 are considered.

D. RBE System Workflow

1) **Setup:** System Administrator (SA) will generate a master secret key mk = (s,k,h) (mk is a vector/tuple of these three values) and a public key pK = (w, v, g^k, g, g^s, …, $(g^s)^q$)
Where w = h^s and v = ê(g, h) and q is the maximum number of users that each role can have and the maximum depth of the role hierarchy.
2) **Manage Role** (mk, ID_R, PR_R)**:** Also SA takes all the role hierarchy and generates the public role parameters for particular role and store them on cloud. Suppose PR_R is a set of identities {ID_{R1}, ID_{R2}, …, ID_{Rm}} of all the roles from which a role inherited all the properties. SA publish a tuple (A_R, B_R, RUL_R) as a public role parameters in the cloud. RUL_R is a role user list specifying which role has which all users.

$$A_R = h^{(s+H_1(\mathsf{ID}_R))\prod_{i=1}^{m}(s+H_1(\mathsf{ID}_{R_i}))}, \quad B_R = A_R^k$$

3) Now suppose owner wants to add or remove a particular user from a role. Then it will send a role id and a user id to role manager which performs the task of adding or revoking a user from a Role User List (RUL_R).

Add User (pk, sk_{Ri} RUL_{Ri}, ID_{Uk}): In this role manager ask for sk_{Ri} to a system administrator. SA computes sk_{Ri} and returs it to Role Manager (RM).

$$\mathsf{sk}_R = g^{\frac{1}{s+H_1(\mathsf{ID}_R)}}$$

Now suppose RM wants to add user U_K with id ID_{UK} in a role R_i with user role list RUL_{Ri}. First it ask cloud to compute the value

$$Y_i = g^{(s+H_1(\mathsf{ID}_{U_k}))\prod_{j=1}^{n}(s+H_1(\mathsf{ID}_{U_j}))}$$

Let suppose Y_i' is the value of Y_i which is previously received from the cloud. If RM is receiving the value for Y_i for the first time then Y_i' = g.

Now to add a user in a role user list RM verifies the below equation.

$$\hat{e}(Y_i', w \cdot h^{H_1(\mathsf{ID}_{U_k})}) \stackrel{?}{=} \hat{e}(Y_i, h)$$

If the equation is satisfied then only RM adds the user to RUL_R. If RM is adding user for the first time for that particular role then it selects two random value r_i , t_i from Zp^*. If some users are already added in a role user list the it uses the same old values of r_i , t_i for the computation. It computes following terms.

$$K_i = v^{r_i}, \quad T_i = g^{-t_i}, \quad W_i = w^{-r_i},$$
$$V_i = Y_i^{r_i} = g^{r_i \cdot (s+H_1(\mathsf{ID}_{U_k}))\prod_{j=1}^{n}(s+H_1(\mathsf{ID}_{U_j}))},$$
$$S_i = H_2(K_i) \cdot \mathsf{sk}_{R_i} \cdot g^{kt_i} = H_2(v^{r_i}) \cdot g^{\frac{1}{s+H_1(\mathsf{ID}_{R_i})}+kt_i}$$

After computing RM add the user id in RUL_R and sends a tuple (T_i, W_i, V_i, S_i) to a cloud. And basis of these values cloud publish another set of role parameters (ID_{Ri}, W_i, V_i, S_i, RUL_{Ri})

Revoke User (pk, sk_{Ri} RUL_{Ri}, ID_U): Now suppose a RM wants to remove a particular user U_k from a Role R_i, then first it directly removes that ID_{Uk} from the RUL_{Ri} and then sends the ID_{Uk} to cloud to compute values.

$$Y_i = g^{\prod_{j=1, j\neq k}^{n}(s+H_1(\mathsf{ID}_{U_j}))}$$

Let suppose Y_i' is the value which is previously received from the cloud. Then RM verify the below equation.

$$\hat{e}(Y_i', h) \stackrel{?}{=} \hat{e}(Y_i, w \cdot h^{H_1(\mathsf{ID}_{U_k})})$$

If the equation hold true RM choose two random values r_i', t_i' from Zp^* and re computes

$$K_i = v^{r_i'}, \quad T_i = g^{-t_i'}, \quad W_i = w^{-r_i'},$$
$$V_i = Y_i^{r_i'} = g^{r_i' \cdot \prod_{j=1, j \neq k}^{n} (s + H_1(\mathsf{ID}_{U_j}))},$$
$$S_i = H_2(K_i) \cdot \mathsf{sk}_{R_i} \cdot g^{kt_i'} = H_2(v^{r_i'}) \cdot g^{\frac{1}{s + H_1(\mathsf{ID}_{R_i})} + kt_i'}$$

RM then replaces the old parameter with a new values of (T_i', W_i', V_i', S_i') in the cloud.

4) **Encrypt** (pk, pub_{Rx}): Suppose an owner wants to encrypt the Message (M) using some specific role R_x. Then it selects any random value z from Zp^*. It ask for the role public parameter of that particular role to a cloud and a public key which is stored on cloud by a system administrator. And it computes following cipher text parts.

$$C_1 = w^{-z}, \quad C_2 = A_{R_x}{}^{z}, \quad C_3 = B_{R_x}{}^{z}, \quad K = v^z$$

Owner uses the above computed key K as an encryption key to encrypt the message M. It uploads the cipher text together C={C1, C2, C3} to the cloud. Also the encrypted data is stored on the cloud so that other user can access it.

5) **Decrypt** (pk, pub_{Rx}, dk_{Uk}, C): Assume that there is a Role R_i which inherits the properties from role R_x. There is a set of users {U_1, U_2,….,U_n} which belongs to the role R_i. Now let user U_1 having role membership of R_i wants to decrypt the Cipher text C which is encrypted by role R_x. Then first user will ask for the Cipher text to the cloud.

$$D = \hat{e}(T_i, C_3), \quad g^{p_{i,\mathcal{M}}(s)}, g^{p_{k,\mathcal{N}}(s)},$$
$$Aux_1 = \prod_{j=1, j \neq i}^{m} H_1(\mathsf{ID}_{R_j}), \quad Aux_2 = \prod_{j=1, j \neq k}^{n} H_1(\mathsf{ID}_{U_j})$$

where

$$p_{i,\mathcal{M}}(s) = \frac{1}{s} \cdot \left(\prod_{j=1, j \neq i}^{m} (s + H_1(\mathsf{ID}_{R_j})) - \prod_{j=1, j \neq i}^{m} (H_1(\mathsf{ID}_{R_j})) \right)$$
$$p_{k,\mathcal{N}}(s) = \frac{1}{s} \cdot \left(\prod_{j=1, j \neq k}^{n} (s + H_1(\mathsf{ID}_{U_j})) - \prod_{j=1, j \neq k}^{n} (H_1(\mathsf{ID}_{U_j})) \right)$$

Cloud will compute the following parameters and sends a tuple and cipher text to user U_1.

$$(C_1, C_2, D, g^{p_{i,\mathcal{M}}(s)}, g^{p_{k,\mathcal{N}}(s)}, Aux_1, Aux_2)$$

Also user request to a system administrator for a user secret key. SA computes the user secret key based on the user Id and returns it to the user.

$$\mathsf{dk}_U = h^{\frac{1}{s + H_1(\mathsf{ID}_U)}}$$

With the above cipher text and parameters received from cloud and user secret key received from SA, user can evaluate the decryption key for message M. User recovers a symmetric encryption key by evaluating below parameter values.

$$K = (\hat{e}(g^{p_{i,\mathcal{M}}(s)}, C_1) \cdot \hat{e}(S_i \cdot H_2(K_i)^{-1}, C_2) \cdot D)^{\frac{1}{Aux_1}}$$

where

$$K_i = (\hat{e}(V_i, \mathsf{dk}_{U_k}) \cdot \hat{e}(g^{p_{k,\mathcal{N}}(s)}, W_i))^{\frac{1}{Aux_2}}$$

Using the key K, User U_k can decrypt the cipher text of Message M and recovers the original data M.

We can see that while evaluating the decryption parameters it requires expansion of two polynomials of m (the total number of ancestor roles) and n (total number of users in a role). This computation of polynomial may require large time. As in these polynomial there are no any private/secret keys associated we can send this computation to a cloud. It will really help to reduce the workload of a user to compute such polynomials for generating decryption key. Also the decryption time is also reduced by a large amount while evaluating the decryption key. Using this approach we can avoid sending full identity user list and a role list to a user end.

III. Implementation

We have implemented the above RBAC system using Java 1.8. We have used Apache Ant 1.9 to auto build our project. The multi-client RBAC model is developed using Java RMI implementation for handling concurrent request to a system administrator. Also as the system interacts over the network RMI helps to communicate with different machines. As we have implemented it on a small scale System Administrator is acting as a role Manager.

The above implementation of RBAC model is done using JPBC 2.0 library which uses concept of bilinear pairing. We have used type "A" curve of a JPBC library which provides Base Field size of 512 bits and Dlog Security values of 1024 bits. Good for cryptosystems where group element size is not critical. Uses supersingular curve Y^2 = X^3 + X. Group order is a Solinas prime[2].

The hashing functionality is implemented using SHA-1 and SHA-256 which gives 160 bit and 256 bit hashed values respectively. SHA-1 and SHA-256 are one way hashing functions.

In our implementation we have used AES technique for encryption and decryption of the input files. The AES symmetric key encryption algorithm uses the key generated by RBE encryption algorithm, and same key is used for decryption which is computed by RBE decryption algorithm.

For input data we have developed a python utility which is used to download Instagram public images based on their location co-ordinates. This utility will download images around 50,000 meters area of that location. Different size images are downloaded to test various file size images encryption and decryption.

IV. Experimental Evaluation

The above implementation is performed on Amazon Aws t2.micro instances. This instance is having single core and 8GB ram. The experiments are performed for varying file size and varying number of users in a role. We have considered 4 ancestor roles for all the experiments. SSH is used to connect to the instance and PSCP is used for transferring files and code on the instances. We used three instances and deployed our code separately for system admin, owner and user. Once the system administrator is active the owner and user can separately connect the system administrator. Below is the explanation of the experiments performed.

The fig 3 shows the comparison of the time taken by encryption and decryption using AES technique. X-axis shows the varying input size of files and Y-axis shows the operation time in milliseconds for encryption and decryption. Until the input file size is 100kb the time taken for both the operations remains same and is very less. However as the file size increases for 1MB the operation time gradually increases for both operations, but in comparison to each other it remains almost same. As the file size increases further the encryption time taken is less as compared to decryption time. Thus it concludes that for small file size the AES encryption and decryption time remains same and vary small till 100Kb file size and after that it increases exponentially.

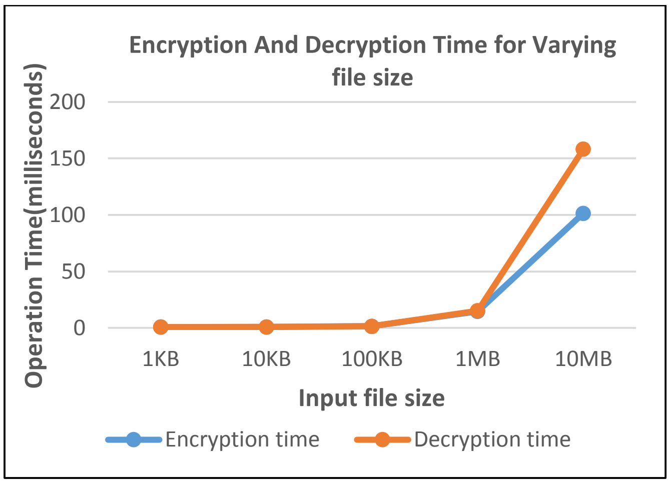

Figure 3. Encryption and Decryption Time for Varying file size

Fig 4 shows the encryption/decryption key generation performance based on our implementation. The X-axis shows the number of users in the role in which the user is trying to decrypt the encrypted images. Y-axis shows the operation time in milliseconds for encryption and decryption. This experiment was performed considering the input file size of 1KB and the role hierarchy is considered as 4 level ancestor roles for this experiments. The graph shows that the encryption and decryption time remains almost same from 10 users to 1000 users. If the trend is seen then the time for encryption algorithm is less as compared to decryption algorithm for all cases. This concludes that even if the number of users increases in a role that has the user who wants to decrypt the image then also the encryption and decryption key generation algorithm operation time remains constant without any major impact on computation time.

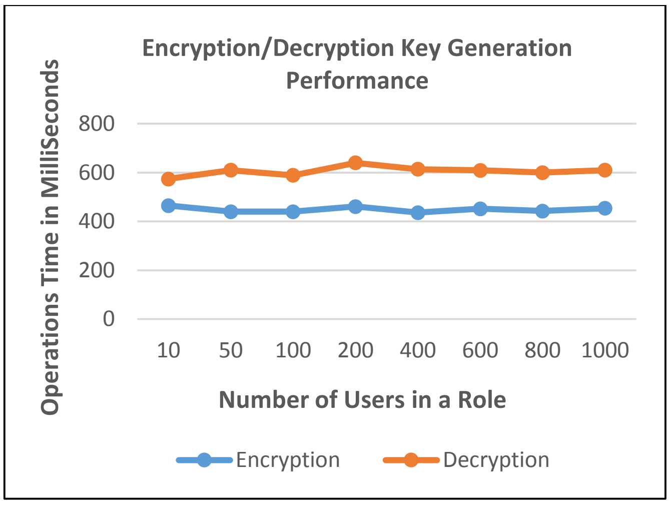

Figure 4. Encryption and Decryption Key Generation Performance

Fig 5. Shows the evaluation result for the role management in our implementation. Role management includes adding new users to a particular role and maintaining the hierarchy. X-axis shows the increasing number of users in a role that are being added at a time. Y-axis shows the operation time in milliseconds for adding users to a role. The graph shows that as the number of users increases from 10, 50 up to 1000 the time taken for adding the user's increases linearly and highest time take for 1000 users is around 120milliseconds. This shows that through our implementation even if 1million users will be added in some seconds or minutes, so our system is scalable for large number of users.

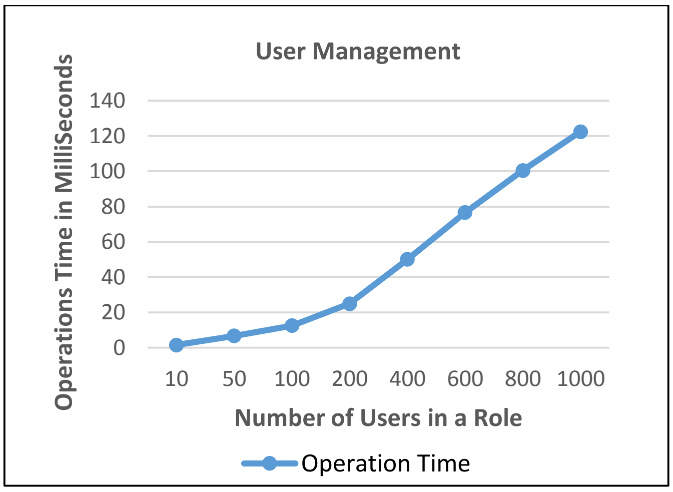

Figure 5. User Management Evaluation Graph

V. Challenges Faced

While implementing we have faced many issues in our project. The JPBC library was complete new to us and we faced issues in implementing these formulae. There was issues in transferring JPBC elements over the network as these elements are not serializable. For this scenario we have converted all the elements in to a byte array and then send them over the network. Also whenever we perform some operations on a JPBC element we have to clone the object, as these elements are mutable. For this we have created a method which was returning a clone for the required element. Also while implementing we are not able to covert the generators G1 and G2 values to a BigInteger values.

We have created a utility which was used to download several images from an Instagram. For varying different file size we had to change the code for different file size as our main code was to download images based on location co-ordinates. For this we had to search for addition terms in Instagram API which allow us to download images of varying size. For connecting to an Instagram API we also need valid credentials which helps in connecting to an Instagram and download images. Also it only allows 5000 Images per hour. We were not able to get images of file size 10MB, so for evaluation purpose of this file size we have considered text files.

VI. Conclusion and Future Work

In this project we see that why there is necessity of an access control system and how it handles role key management issue. This system helps users to store and access data securely from the cloud. We have seen the role based encryption scheme and used it in implementing cryptographic RBAC system using JPBC library. It’s a secure hybrid cloud storage system architecture which helps to provide a security to a private or a sensitive data of an organization. It helps to store the sensitive data on the private cloud, while storing all sharable data on public cloud. As all computation such as computing encryption / decryption parameters is done on cloud, it reduces the computational cost on a large scale which is required on client side. As per the section IV we have seen that as the number of user increases in a particular role, the computation time required for generating Encryption / Decryption key remains almost same. Thus making the system more scalable. As the system is on cloud, its performance can be increased at any time by changing the instance type as required by an organization.

Even though the RBAC models provides a security in access the data, there are some trust issues in Role-Based Access Control system. And due to this there is necessity of Trust Enhanced Cryptographic Role-Based Access control model. So the trust models addresses the missing part of trust in RBAC scheme to store data securely on cloud. Also it provides more robust and secured storage system than using cryptographic approaches alone. There are two trust models to assist the above trust issues in RBAC. (i) Owner-Role RBAC trust model and (ii) Role-User RBAC trust model. In future we can implement these models into our RBAC system to handle trust issues.

REFERENCES

[1] Lan Zhou, Vijay Varadharajan, and Michael Hitchens "Achieving Secure Role-Based Access Control on Encrypted Data in Cloud Storage" in IEEE Transactions on Information Forensics and Security, VOL. 8, NO. 12, DECEMBER 2013

[2] http://gas.dia.unisa.it/projects/jpbc/#.VxmP0ubQOro

[3] https://docs.oracle.com/javase/tutorial/rmi/

[4] https://www.instagram.com/developer/