SECURE DATA SHARING IN CLOUD

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Engineering and Architecture

By

Revathi Dhamotharan

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Electrical and Computer Engineering

October 2014

Fargo, North Dakota

North Dakota State University
Graduate School

Title

Secure Data Sharing in Cloud

By

Revathi Dhamotharan

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Samee U.Khan
Chair

Dr. Jacob Glower

Dr. Na Gong

Dr. Saeed Salem

Approved:

11/17/2014	Dr. Scott C.Smith
Date	Department Chair

ABSTRACT

Cloud storage is widely used for sharing data due to low cost maintenance. But, it is also necessary to secure data on cloud. Secure Data Sharing in Cloud focuses mainly on: a) privacy and confidentiality, b) key management and encryption, c) secure data sharing without re-encryption and d) forward and backward access control. When user wants to share data, it sends request to trusted party that generates a symmetric key which is used to encrypt the data for sharing. This key is used to compute two key shares for trusted party and user. The key is deleted using secure overwriting. Its working was formally verified using High Level Petri Nets, SMT Library, and Z3 solver. It was implemented in Visual Studio and its performance was evaluated based on time consumption for various operations which revealed that it has the potential to be effectively used for secure data sharing in cloud.

ACKNOWLEDGEMENTS

Firstly, I would like to thank my advisor, Dr. Samee U. Khan for his immense support, patience and guidance throughout the research work. I am grateful to my committee members, Dr. Jacob Glower, Dr. Saeed Salem and Dr. Na Gong, for their valuable advice and suggestions which have been a great help in completing this work successfully. Most importantly, I am extremely grateful to Mazhar Ali for all his help and guidance during the research.

Last but not the least, I am grateful to my family and friends for their support during my research work and progress towards completion.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Cloud computing can be defined as a model for enabling ubiquitous, convenient and on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort from the user side and minimal service provider interaction [9].It has found its way into a lot of small organizations and individuals as the use of cloud storage has reduced the need to maintain any physical resources. The most commonly used are Amazon S3 and Google cloud. The users just have to pay the fees to access the cloud storage resources. Due to this, data sharing has become easy for individuals as well as organizations. They are able to access their data from anywhere anytime without the fear of losing data. But, the use of cloud computing comes along with a variety of issues of its own like loss of control over data, security, privacy and confidentiality. Hence, there is a need to secure data not only from unauthorized users but also from the cloud provider. As the resources are shared, the users need to manage the use of cloud storage to reduce these risks. They need to make sure that data stored on the cloud is accessed only by members of the group to reduce the risk of losing control over data. Apart from this, the users also need to make sure that the data is being stored and shared securely in the cloud.

One of the widely used options to secure data is to encrypt data using cryptography which provides a wide range of methods to encrypt data and ensure its privacy and confidentiality. Encrypting data before uploading it to the cloud assures privacy from the cloud service provider well. Cryptographic methods involve key management, encryption and decryption processes.

Key management takes care of key generation for all the users including the new users that may join later. If, all the members of a group use a single key, the users need to be more careful with key management. Key management should be able to cope up with the frequent changes in membership as it may result into high computation overhead if not handled properly. A simple solution would be rekeying. So, when a new user is added, the group generates a new key, re-encrypts the data and distributes the keys again to all the members. On the other hand, when a member leaves the group, the group performs the same procedure of rekeying to avoid the departing users from using the current key to access data illegally. But, this results in a lot of overhead in case of frequent changes to the group membership.

Apart from this, there is always a threat from malicious authorized users within the group who may try to access and tamper the data. In case of a single symmetric key, if users of the group hold the entire key, it may prove to be a serious threat if any of the authorized users turn malicious. Though data is encrypted, it is thus necessary to handle key management efficiently to provide secure data sharing in cloud storage.

In SeDaSC, the proposed methodology involves mainly three entities:

a) Cryptographic Server
b) Cloud Storage
c) Users

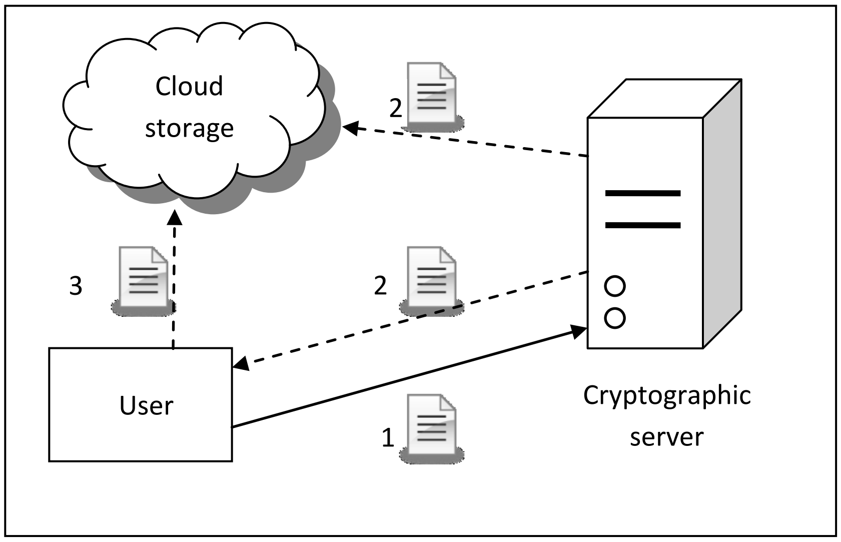

Figure 1. Basic concept behind SeDaSC

The cryptographic server takes care of access control, key generation, encryption and decryption processes. Access control gives all the information about the users including their rights, permissions, joining date, etc. Key generation involves generating a single symmetric key which is used to encrypt the data. This key is then split into two key shares for each user in the group. One key share stays with the cryptographic server in access control and the other key share is sent to each corresponding user and the original key is deleted using secure overwriting. When a new user is added, the original key is generated again using owner's key shares. This is, in turn, used to generate the key shares for the new user. The key share for cryptographic server is added to the access control list along with the rest of the information about the new user. While the user key share is sent to the new user. On the other hand, when a user departs from the group, the cryptographic server just needs to delete all the information related to this user. In some methods, a single encryption key is shared between all the users. In such cases, when a user is deleted, there is a risk that the user may turn into a threat as it still has access to the complete encryption key. This is avoided by generating the encryption key again and re-encrypting the data. But, frequent changes in the number of users may result in computation overhead. These issues have been addressed in SeDaSC by efficiently handling key management without using re-encryption.

The cloud provides storage services to the users. The data file is encrypted by the cryptographic server and uploaded to the cloud. Whenever a user requests access to this data file, the cryptographic server or the recipient user downloads the encrypted file from the cloud, validates this user, decrypts the data and sends it back to the user based on the user rights.

Users act as client to the cloud. The data file is owned by one of the users and is shared with other users. The data owner decides who can access and edit the file. The major contributions through this proposed methodology are as follows:

- Confidentiality of data by using symmetric encryption.
- Secure data sharing over the cloud without the use of Elliptic Curve or Bilinear Diffie-Hellman Problem (BDH) cryptographic re-encryption.
- Secures data from malicious users within the group due to the possession of only a part of the symmetric key.
- Handles issues related to forward and backward access control due to insider threats.
- Formal modeling and verification of SeDaSC using High Level Petri Nets (HLPN), SMT-Lib and Z3 Solver.

The design of the proposed scheme and its working will be explained further in the later chapters.

2. RELATED WORK

There has been a growing interest in the study of cloud storage and ways to secure the data. One of the major concerns here is to control which users have access to data. Some methods use access control to manage which users have the rights to read/write data stored on cloud. But, access control may not be enough to secure data on cloud. An alternate solution to this is encryption. Data is encrypted by the owner before submitting it to the cloud. This secures data from both cloud service provider as well as unauthorized users. Encryption can be of two types- symmetric and asymmetric. Symmetric encryption uses the same key to encrypt and decrypt the data. Unlike symmetric encryption, asymmetric encryption uses two keys- public and private key. The owner encrypts data with the recipient user's public key but this encrypted data can be decrypted using only the recipient user's private key. This helps in securing the data from unauthorized users. But, when a user leaves the group or is revoked by the owner, the user can still decrypt the data as it still possesses the private key that can be used for decryption. In order to avoid this, the owner has to generate the keys again and re-encrypt the data. This results in a major computation overhead in case of frequent changes to the members of the group.

Lei et al. [5] proposed a Certificateless Proxy Re-encryption scheme where the data owner first encrypts the data using symmetric data encryption key before uploading it to the cloud. The owner then generates proxy re-encryption keys along with all the users. The cloud uses these re-encryption keys to transform the data stored in the cloud and sends it to the recipient user. This transformed data can then be decrypted back to its original form using the recipient user's private key. By using the symmetric data encryption key, this scheme ensures that the cloud does not have any access to the original data. The proxy-re-encryption in this scheme is based on bilinear pairing and BDH which results in high computational overhead.

In order to overcome this high computational overhead, the authors in [2] proposed a mediated certificateless encryption scheme that does not use pairing operations. The data is encrypted using the users' keys generated by the cloud based on the access control policies. When a user requests this data, the cloud decrypts the encrypted data partially which can be decrypted to the original data using the user's own private key. Since, the cloud handles partial decryption and key management, user revocation is easier to handle. But, this might prove to be a risk from security point of view. In addition to this, as decryption is performed twice, it increases the computation cost of decryption which in turn reduces the advantages of using pairing free approach.
The authors in [1] proposed a scheme that used El-Gamal cryptosystem and bilinear pairing to share data securely on the cloud. This scheme was based on incremental version of cryptography where data is divided into blocks and each block is encrypted incrementally. This incremental re-encryption is handled by a trusted third party which acts as a proxy and manages both encryption and access to the data. In spite of these advantages, the computation overhead due to bilinear pairing still exists.

The authors in [11] proposed a methodology based on the shared key derivation method for securing data sharing among the group. This methodology uses binary tree for key computation. But, the computational cost for this scheme is high due to the rekeying mechanism. Apart from this, the scheme is not tailored for the public cloud system because certain operations require centralized mediations.

SeDaSC methodology proposes a scheme based on symmetric encryption which helps in sharing data securely in the cloud without using El-Gamal cryptosystem, BDH and bilinear paring. In addition to privacy and confidentiality from both cloud service provider and malicious users, this scheme also handles forward and backward access control as each user has access to only a part of the key and not the complete key.

3. SECURE DATA SHARING

This chapter discusses the proposed methodology that can be used to secure data stored on the cloud without using re-encryption and its working. This chapter also presents the entities involved in this methodology, key generation and the design for this methodology.

3.1. Entities

This methodology involves majorly three entities:

a) The Cloud - This entity provides storage services to the users. In order to preserve the privacy and confidentiality of data, it is encrypted before being uploaded to the cloud. This also ensures that the data is secured from the cloud service provider. In this methodology, the cloud is just involved in the basic operations where data is uploaded and downloaded from the cloud storage. This reduces the overhead on the cloud. So, when a user requests access to the data file, the data file is just downloaded from the cloud, decrypted to the original form and sent back to the user based on the user rights.

b) Cryptographic Server - This trusted entity handles all the operations required to secure the data stored on the data. When the owner wants to share some data with a group, it sends data to the cryptographic server. The cryptographic server then encrypts data and sends it either to the owner to upload it to the cloud or sends it directly to the cloud for storage. The cryptographic server also maintains the Access Control List (ACL) which is initially sent to the cryptographic server as a list of users by the owner along with the file. ACL provides all the information about all the members of the group and their rights to access the file.

c) Users - These act as client to the cloud storage. When the owner wants to share some data with a group. The owner decides who can access and edit the file. As described earlier, the owner sends all this information in the form of ACL to cryptographic server along with the file to be shared. It also controls all the user rights and the changes to the membership of the group.

3.2. Cryptographic Keys

SeDaSC uses a single symmetric key for securing data. But, in order to avoid unauthorized access by revoked members, this key is split into two key shares and the original key is deleted using secure overwriting after successful encryption/decryption. One of the key shares stays with the cryptographic server while the other key share is sent to the corresponding user. The symmetric key and the key shares as generated in the following way:

a) Symmetric Key (K) – This is a random symmetric key generated by the cryptographic server. K is generated to be 256 bits which has been recommended by most of the standards for symmetric algorithms [10][10].First, a random 256-bit number(R) is generated such that R = $\{0, 1\}^{256}$. This R is then provided as an input to a hash function (H_f). H_f could be any hash function with a 256-bit output. This method uses SHA-256 as H_f which takes R as input and generates 256-bit symmetric key, K. K is then used by a symmetric algorithm which is AES, in this case, for both encryption and decryption. K is then split into two key shares.

b) CS key share, K_i -The cryptographic server generates a K_i= $\{0, 1\}^{256}$ for each user. The cryptographic server makes sure that K_i is different for each user. This K_i stays with the cryptographic server and is stored in the ACL for later use.

c) User key share, K_i' - K_i' is computed for each user in the following way:

$$K_i' = K \oplus K_i \tag{1}$$

This is computed for every user and sent to the corresponding user which is then used to generate K when the user needs to access the file. Once, the key shares have been generated and the encryption/decryption operation has been carried out, the symmetric key K is deleted by using secure overwriting.

Algorithm 1 Key generation and encryption

```
Input:
$F, ACL$, Symmetric Key Algorithm $SKA$, 256-bit Hash Function $H_f$
Compute:
$R = \{0, 1\}^{256}$
$K=H_f(R)$
$C=SKA(F, K)$
for each user $i$ in ACL do
    $K_i = \{0, 1\}^{256}$
    $K_{i'} = K \oplus K_i$
    Add $K_i$ for user $i$ in ACL
    send $K_i'$ for user $i$
end for
$delete(K)$
$delete(K_i')$
return $C$ to the owner or upload to the cloud
```

Figure 2. Algorithm for key generation and encryption

3.3. SeDaSC Design

This section presents the design of the methodology.

3.3.1. File Upload

When the owner wants to share some data with some users, the owner sends a request to the cryptographic server. This request includes the data file, the list of users (L) that would have access to this file. L contains all the information about the users like their group id, user id, access rights and other necessary information required to control their access. Upon receiving the request, the cryptographic server generates a group id for this group if it does not already exist. If the group does not exist already, it also generates an Access Control List (ACL) using L which is maintained by the cryptographic server. It then generates the single symmetric key K, K_i and K_i**'** for each user. K is used to encrypt the file using an appropriate symmetric key algorithm (AES in this case). As discussed earlier, K_i stays with the cryptographic server and is inserted into the ACL for later user while K_i**'** is sent to each corresponding user along with the encrypted file and group id (if this is a new group) over a secure communication channel. The key K is deleted by secure overwriting once the key shares have been generated and data has been encrypted. To protect the integrity of the file, the cryptographic server also computes the HMAC signature on every encrypted file. A similar procedure for the HMAC key is adopted. However, HMAC key is kept with the cryptographic server only. The encrypted data can be handled in two possible ways. It can be sent to the owner over a secure communication channel that can then upload it to the cloud for sharing. On the other hand, the encrypted file can be directly uploaded to the cloud by the cryptographic server. Figure 3 shows the file upload operation and Figure 2 shows the process for key generation and encryption.

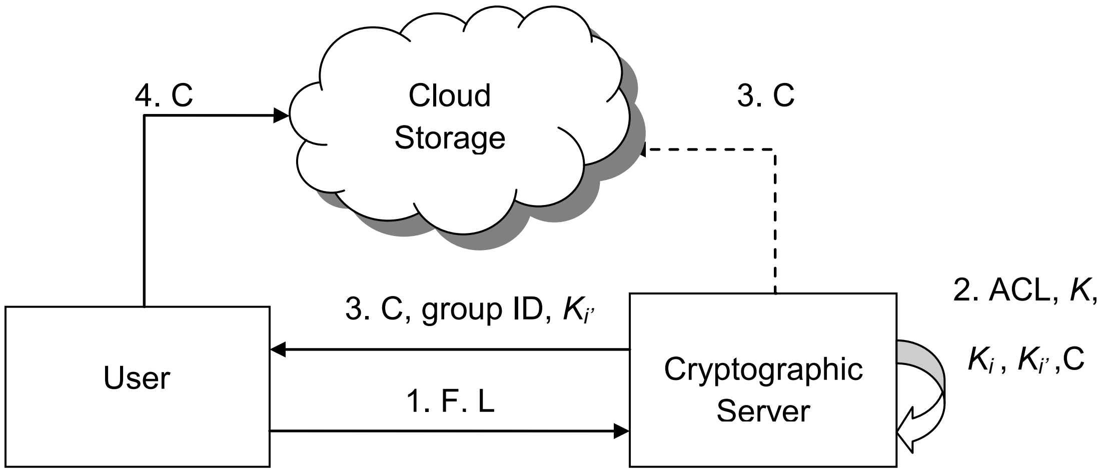

Figure 3. File upload

3.3.2. File Download

When a user wants to access the file, it either sends the request to the cryptographic server or downloads the encrypted file from the cloud and sends it to cryptographic server along with its key share, K_i' . The cryptographic server then validates the user and its rights using ACL. If the user has the right to access the file, the cryptographic server then computes the symmetric key K by applying Xor to K_i and K_i' corresponding to the recipient user. This key K is then used to decrypt the file. If the key K is correct, the file is decrypted successfully. After successful decryption, this file is then sent to the user over a secure communication channel which can be achieved using SSL or IPSec and the key K is deleted by secure overwriting. As mentioned earlier in this section, the encrypted file can be either downloaded by the user and sends it to the cryptographic server or the cryptographic server can directly download it from the cloud (as shown in Figure 5). Figure 4 shows the working of the decryption process. Figure 5 and Figure 6 show the file download operation.

Algorithm 2 Decryption algorithm

```
Input:
$C, ACL$, Symmetric Key Algorithm $SKA$
Compute:
Get $K_i$' from the requesting user
Get $C$ from the requesting user or download from the cloud
Retrieve $K_i$ from the $ACL$
If $K_i$ does not exist in the $ACL$ then
    return access denied message to the user
else
    $K = K_i \oplus K_i'$
    $F = SKA(C, K)$
    send $F$ to the user
end if
$delete(K)$
$delete(K_i)$
```

Figure 4. Algorithm for decryption

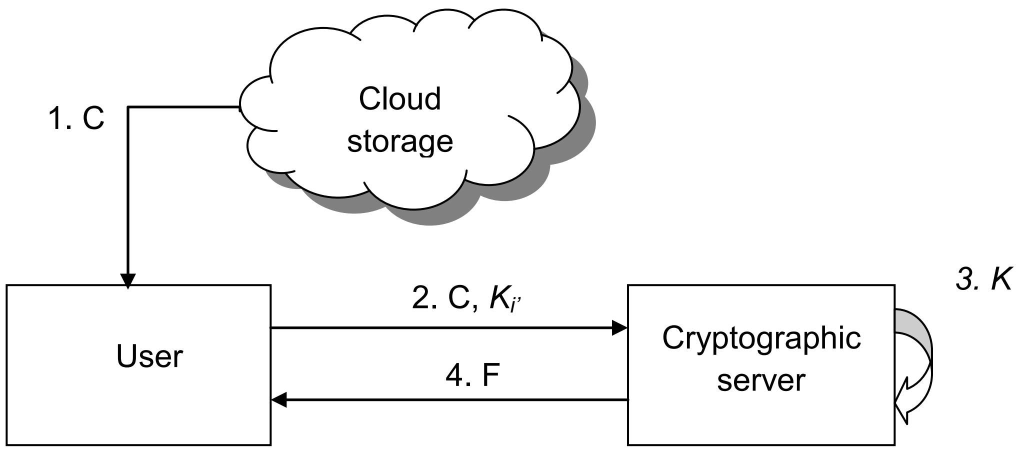

Figure 5. File download

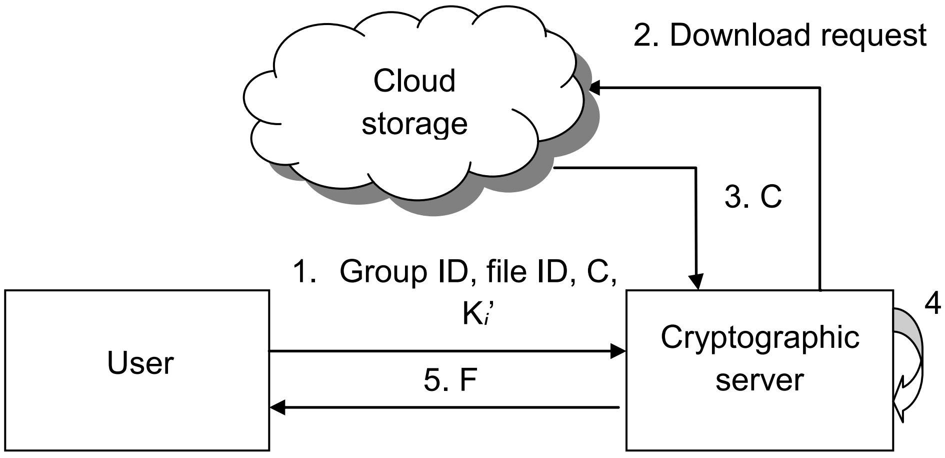

Figure 6. File download (special case)

3.3.3. File Update

When a user wants to update/edit the file, it sends a request to the cryptographic server or downloads it directly from the Cloud. The request includes the encrypted file, group id and K_i**'**. Upon receiving the request, the cryptographic server validates the user and uses ACL to check if the user has rights to access and edit the file. If the user doesn't have sufficient rights, the cryptographic server denies the user's request.

Otherwise, the cryptographic server performs the same procedure as in downloading the file and sends the decrypted file to the user over a secure communication channel. Once the user receives the file, it can edit the file. The user can then upload this file to the cloud using the same process carried out while uploading the file to the cloud. But, this won't involve key generation for each user and ACL creation as both already exist. It only involves encrypting the file using K which is computed by applying Xor to K_i and K_i'. The key K is deleted as always using secure overwriting. The complete file update process is shown in Figure 7.

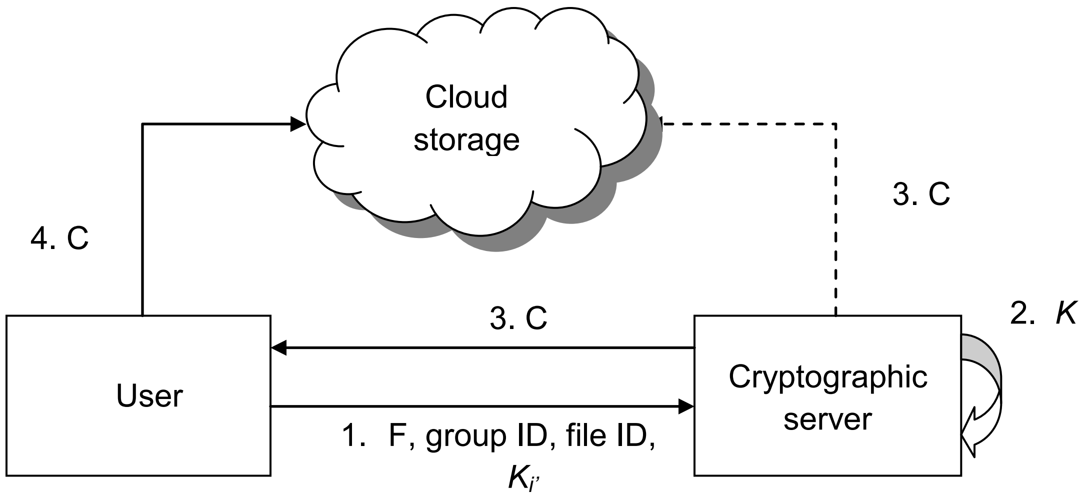

Figure 7. File update

3.3.4. New Group User Inclusion

When the owner needs to add a user to the group, it sends a request to the cryptographic server along with the group id, user id, joining date and other information related to access rights and permissions. The ACL may also contain information if the user has access to old data. This ensures backward access control for the joining member. The cryptographic server updates ACL with all the information for this new user.

3.3.5. User Deletion

When the owner wants to delete a user, the owner just needs to send a request to the cryptographic server along with user id and group id. On receiving the request, the cryptographic server looks for the user information in ACL and deletes all the information related to this user. As the user just has its own key share and not the whole key, the user would no longer be able to access the file and decrypt it which ensures Forward Access Control.

3.4. Advantages of SeDaSC

The SeDaSC is proposed to provide the following services to the outsourced data:

- Confidentiality;
- Secure data sharing among the group;
- Secure data from unauthorized access of valid insiders within the group and
- Forward and backward access control to counter insiders and departing group users.

The following discussion briefly describes how above mentioned services are achieved.

The cloud is not considered to be a secure and trustful entity in the context of SeDaSC. Multi tenancy, virtualization, and shared pool of resources may pose many forms of insider and other threats to the data. Moreover, the cloud may also retain copies of the file even after it is requested for deletion.

In case of SeDaSC, the file is encrypted with K. The K is generated at the cryptographic server and is deleted right after utilization using secure overwriting. The cryptographic server or the user cannot reconstruct K all by themselves. For confidentiality, the data cannot be leaked unless the attacker gains access to K. The K in its entirety is not stored anywhere, neither it travels on the communication channel. Therefore, the access to K is a difficult task. Even if an attacker gets hold of user share, K_i', he will have to guess the other share correctly. The guess or random generation is to be made from total of 2^256-1 possible shares. The probability of generating the correct share is 1/ (2^256-1) =8.636 × 〖10〗^ (-78) which is negligible. Moreover, if the insider within the cloud gets access to the file, the absence of K will be a barrier to subvert the confidentiality of the data.

For secure data sharing, SeDaSC does not utilize the concept of re-encryption with multiple keys. The encryption is done with a single symmetric key. However, the authorized users are granted access on the basis of possession of the key share and the typical authentication and authorization phenomenon. The ACL lists the authorized users with their credentials and corresponding cryptographic server key shares. After user authentication, the user key share is used along with cryptographic server share to generate K. As the user key share is possessed by only a valid user, only a valid user can lead to a successful encryption/decryption of the data.

The division and dispersal of the key also helps to counter the insider malicious users within the group. The ACL is maintained separately for each group file. Therefore, a valid group user cannot access the group file that is not shared with him/her. An attempt to access an unauthorized file is also blocked by the fact that the user will not have the key share for that file. Moreover, the ACL of the unauthorized file will not contain any record for this malicious user. Furthermore, the absence of entire key with the user and ACL collectively ensure the forward and backward access control for the data.

Most of the data forwarding schemes are dependent on the El-Gamal cryptosystem and bilinear pairing [1]. The aforesaid schemes require re-encryption of the data each time any user other than the owner requests access to data. The El-Gamal cryptosystem is computationally intensive. Moreover, re-encryption at each access adds to the overhead. SeDaSC methodology utilizes symmetric encryption and access by multiple users is achieved through key management as explained in the preceding section. Therefore, the overhead of the SeDaSC methodology is less as compared to the traditional El-Gamal based re-encryption systems.

4. FORMAL ANALYSIS

4.1. Formal Analysis and Verification

Before going into the details of the formal analysis of the proposed methodology, this chapter provides a brief introduction to HLPN, SMT-Lib, and Z3 solver for better understanding.

4.2. High Level Petri Nets

The Petri Nets are used for graphical and mathematical representation of the system. The Petri Nets can model a range of systems, such as distributed, parallel, concurrent, non-deterministic, stochastic, or asynchronous [6]. We have used a variant of conventional Petri Net called High Level Petri Net (HLPN). A HLPN is a 7-tuple structure represented as $N = (P, T, F, \varphi, R, L, M_0)$, where P denotes the set of places, and T refers to the set of transitions such that $P \cap T = \emptyset$. The flow relations are represented by F such that $F \subseteq (P \times T) \cup (T \cup P)$. The φ map places P to the data types. The R defines the set of rules for transitions. The L is a label on F and M_0 represents the initial marking [6]. The information about the structure of the net is provided by (P, T, F) whereas (φ, R, L) provides the static semantics that means the information does not change throughout the system.

4.3. SMT-Lib and Z3 Solver

The SMT is used for validating the satisfiability of rules over the theories under consideration. The SMT has roots in Boolean Satisfiability Solvers (SAT) [7]. The SMT-Lib provides a common input platform and benchmarking framework that helps in the evaluation of systems. We use Z3 solver with SMT-Lib that is not only a theorem prover developed at Microsoft Research but it is also an automated satisfiability checker. In addition, Z3 determines whether the set of formulas are satisfiable in the built-in theories of SMT-Lib. Readers are encouraged to see [8] for the use of SMT-Lib in the verification process.

Table 1. Data types for the HLPN model

Data types for the HLPN model	
Data type	Description
F	A string type holding data to be protected
K	A string type representing a symmetric key
K_i	A string type representing first constituent part of K for user i
K_i'	A string time representing second constituent part of K for user i
C	A string type representing encrypted data
U_i	A number representing i-th user
Gid	A number representing group ID
H	A string representing hash value of the data

Table 2. Places and mappings in the HLPN model

Place	Mapping
φ(User)	$\mathbb{P}$ (F×U_i×K_i×gid ×C× H)
φ (CS)	$\mathbb{P}$ (F ×U_i×gid ×K×K_i×K_i'× C ×H)
φ (Cloud)	$\mathbb{P}$ (C ×H)

Verification is the process that checks for the correctness of the system. Here, it uses the bounded model checking that verifies whether for any input parameters the system terminates after finite number of states or not. In the bounded model checking: **(a)** the description of the system is provided stating properties or rules of the system, **(b)** system is represented by a model, and **(c)** some verification tool is used to check whether the model holds the specified properties or not. The HLPN model for the SeDaSC is shown in Figure 6. The data types and mappings are shown in Table 1 and Table 2, respectively. In HLPN model, all the rectangular black boxes are transitions and belong to set T. The circles are termed as places in the set P. Whenever the data is to be shared among multiple users, the data owner sends the data file, F, to the cryptographic server. The list of users is also sent along with the F and other parameters discussed in chapter 3. The following rule is mapped to the transition *Send_d* of the HLPN:

$$R(Send\_d) = \forall x_1 \in X_1, \forall x2 \in X_2 \mid x_2[1] := x_1[1] \wedge x_2[2] := x_1[2] \wedge X_2' = X_2 \cup \{x_2[1], x_2[2]\} \quad (2)$$

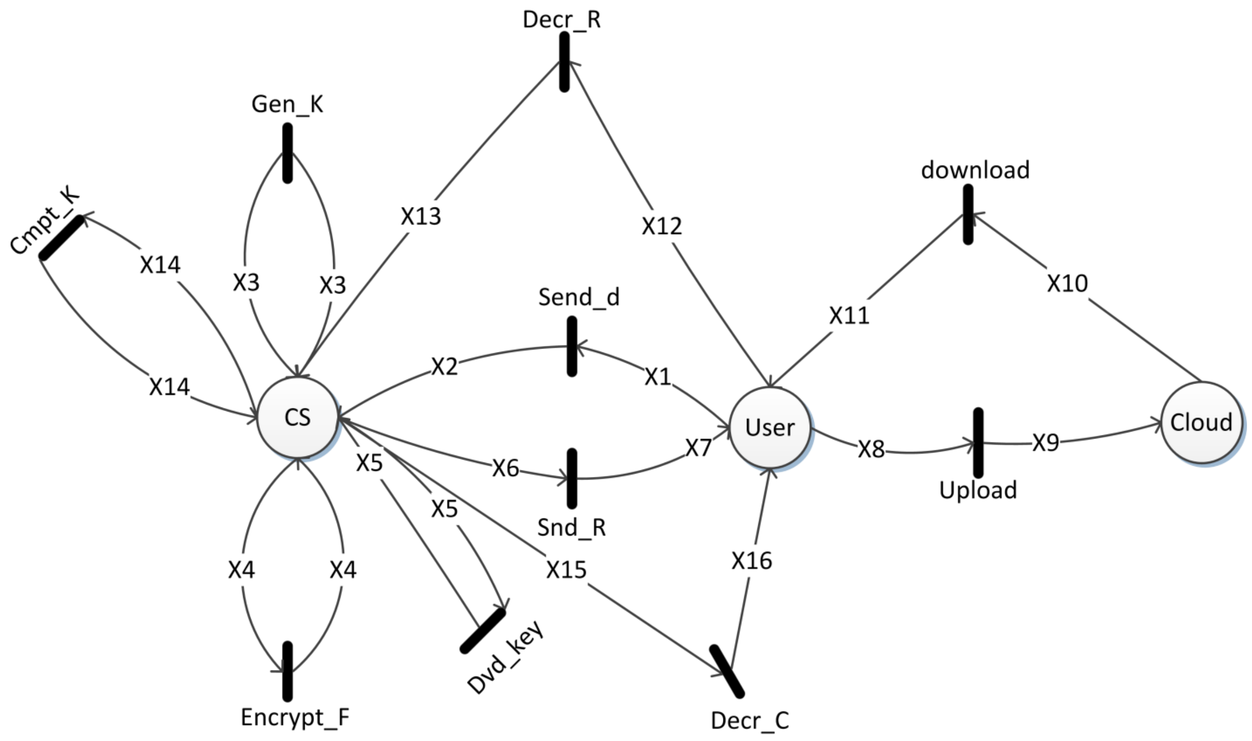

Figure 8. The HLPN model for SeDaSC

The cryptographic server generates a symmetric key, *K*, and other parameters according to the earlier explained procedure. The following formula operates on the transition *Gen_K* to depict the process:

$$R(Gen\_K) = \forall x_3 \in X_3 \mid x_3[3] := gen\_groupID(x_3[3]) \wedge x_3[4] := gen\_K() \wedge$$

$$X_3' = X_3 \cup \{x_3[3], x_3[4]\} \quad (3)$$

The cryptographic server computes the hash of *F* and encrypts the *F* with the symmetric key, *K*. The result is cryptographic data, *C*. The process is carried out at the transition *Encrypt_F* with the following rule:

$$R(Encrypt\_F) = \forall x_4 \in X_4 \mid x_4[8] := hash(x_4[1]) \wedge x_4[7] := encrypt(x_4[1], x_4[4]) \wedge$$
$$X_4' = X_4 \cup \{x_4[7], x_4[8]\} \quad (4)$$

The cryptographic server computes the two constituent shares of the K, K_i and K_i', for each of the users in the ACL and deletes K afterwards. The transition Dvd_K depicts the procedure with the following formula:

$$R(Dvd\_K) = \forall x_5 \in X_5 \mid x_5[5] := gen\_Ki() \wedge x_5[6] := x_5[4] \oplus x_5[5] \wedge over\_write(x_5[4]) \wedge$$
$$X_5' = X_5 \cup \{x_5[4], x_5[5], x_5[6]\} \quad (5)$$

The encrypted data C along with the hash value, group ID, and K_i' is sent to the data owner. The procedure is detailed in chapter 3. The following formula at the transition Snd_R shows the process:

$$R(Snd\_R) = \forall x_6 \in X_6, \forall x_7 \in X_7 \mid x_7[3] := x_6[6] \wedge x_7[4] := x_6[3] \wedge x_7[5] := x_6[7] \wedge x_7[6] := x_6[8] \wedge$$
$$over\_write(x_6[6]) \wedge X_6' = X_6 \cup \{x_6[6]\} \wedge X_7' = X_7 \cup \{x_7[3], x_7[4], x_7[5], x_7[6]\} \quad (6)$$

The user uploads the encrypted data to the cloud. The following rule maps to the transition *Upload:*

$$R(Upload) = \forall x_8 \in X_8, \forall x_9 \in X_9 \mid x_9[1] := x_8[7] \wedge x_9[2] := x_8[8] \wedge X_9' = X_9 \cup \{x_9[1], x_9[2]\} \quad (7)$$

The downloading user downloads the encrypted data from the cloud. The following formula relates to the transition *download*:

$$R(download) = \forall x_{10} \in X_{10}, \forall x_{11} \in X_{11} \mid x_{11}[5] := x_{10}[1] \wedge x_{11}[6] := x_{10}[2] \wedge$$
$$X_{11}' = X_{11} \cup \{x_{11}[5], x_{11}[6]\} \quad (8)$$

The user sends decryption request to the cryptographic server along with C, U_i, gid, and K_i'. The following rule maps to the transition $Decr\_R$:

$$R(Decr\_R) = \forall x_{12} \in X_{12}, \forall x_{13} \in X_{13} \mid x_{13}[2] := x_{12}[2] \wedge x_{13}[3] := x_{12}[4] \wedge x_{13}[6] := x_{12}[3] \wedge$$
$$x_{13}[7] := x_{12}[5] \wedge X_{13}' = X_{13} \cup \{x_{13}[2], x_{13}[3], x_{13}[6], x_{13}[7]\} \quad (9)$$

The cryptographic server after verifying the authorization status of the user from the ACL computes K according to the procedure defined in chapter 3. The following transition and rule shows the process:

$$R(Cmpt\_K) = \forall x_{14} \in X_{14} \mid x_{14}[4] := x_{14}[5] \oplus x_{14}[6] \wedge X_{14}' = X_{14} \cup \{ x_{14}[4] \} \quad (10)$$

The cryptographic server decrypts the data and sends it back to the user. The K and K_i' are deleted subsequently. The transition $Decr\_C$ shows the process:

$$R(Decr\_C) = \forall x_{15} \in X_{15} , \forall x_{16} \in X_{16} \mid x_{16}[1] := decrypt(x_{15}[7] , x_{15}[4]) \wedge$$
$$X_{16} = x_{16} \cup \{x_{16}[1]\} \quad (11)$$

4.4. Verification of Properties

The properties that are verified are the following:

1. A valid user in the group cannot lead to the generation of a valid K by pretending to be another user and contributing a random K_i',
2. A valid user in the group leads to the generation of a valid K by contributing a valid K_i', and
3. A malicious user outside the group, if somehow gets access to the encrypted file, cannot lead to its decryption.

The above given model was translated to SMT-Lib and verified through Z3 solver. The solver showed that the model is workable and executes according to the specified properties. Z3 solver took 0.085 seconds to execute the working of the proposed model.

5. PERFORMANCE EVALUATION

5.1. Experimental Setup

To evaluate the performance of the proposed methodology, it was implemented in Visual Studio 2010 C# using .Net 4 framework. As discussed earlier, the proposed methodology consists of three entities, namely: **(a)** the cloud, **(b)** the cryptographic server, and **(c)** users. The Amazon Web Services (AWS) SDK was used to avail the use of .Net APIs and communicate with Amazon S3 which serves as the cloud server in our implementation. The cryptographic server is implemented as a third party. The functionality required by the user is implemented as a client application that connects with the cryptographic server to manage all the file uploads and downloads in addition to any changes in membership. The hardware characteristics for the cryptographic server and the user client are as shown in Table 3.

Table 3. Hardware characteristics for cryptographic server and user client machines

CPU	Intel(R) Core ™ i3-3217U CPU 2 @ 1.80 GHz 1.80 GHz
RAM	4 GB
Storage	450 GB
OS	Windows 8 64 bits

The proposed scheme needs to make sure that the communication channel used by the users, cloud and cryptographic server are secured by either SSL or IPSec. Hence, the communication between the entities was accomplished using .Net libraries (System.Net.Security,System.Net.Sockets). The classes TcpClient and TcpListener have been used to implement TCP protocol. This communication channel was then secured using the SSLStream class. The scheme uses SHA-256 hash function for generating keys and AES (symmetric encryption algorithm) for encryption and decryption. The scheme was implemented using a .Net library, System.Security.Cryptography. The class SHA256CryptoServiceProvider within library was used to access all of the methods related to SHA256. All of the cryptographic operations, namely: the encryption and decryption were implemented using the AES class that represents the base abstract class for AES algorithm.

5.2. Results

This section presents and discusses all the experimental results for SeDaSC.

5.2.1. Key Generation

As discussed earlier, key generation involves the process of generating the symmetric key along with the cryptographic server and user key share that are computed separately for each user. In this experiment, the time required to generate keys is evaluated for different number of users (10 to 100).

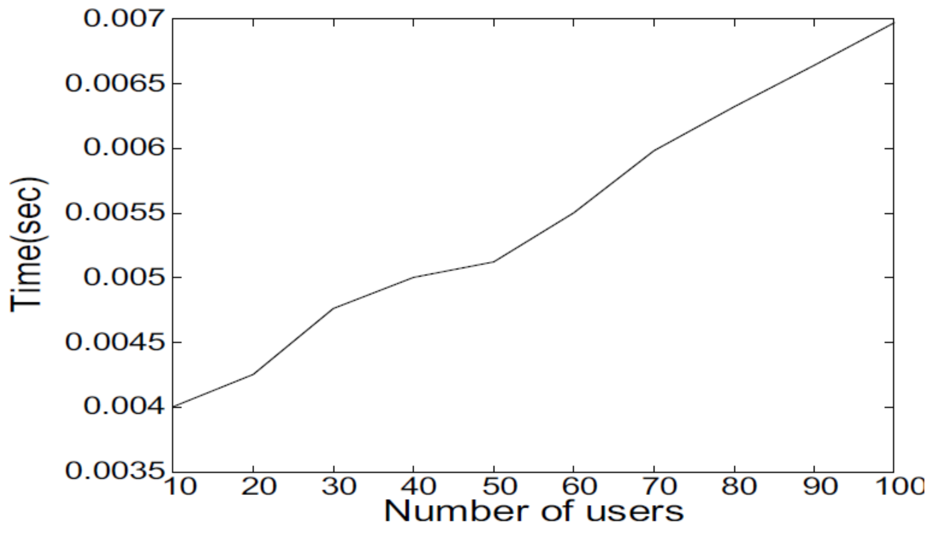

Figure 9. Time consumption for key generation.

The results are shown in Figure 9. As is evident in Figure 9, the time for key generation increases with the number of users. Major part of this time is due to the fact that key shares need to be generated separately for each user. But, the time required to generate the symmetric key remains almost constant and does not depend on the number of users.

5.2.2. Encryption and Decryption

To evaluate the performance of encryption and decryption, experiments were performed to obtain the time required for encryption and decryption for different file sizes. The time for encryption here includes time for key generation and encryption. The results for these experiments are as shown in Figure 10.

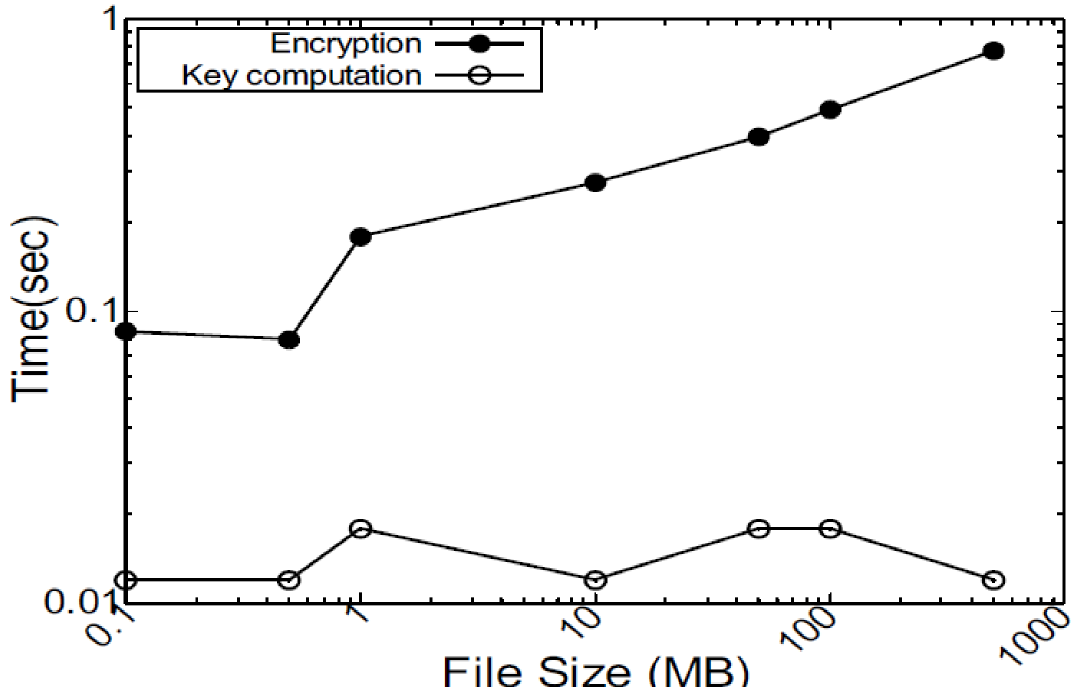

Figure 10. Time consumption for file encryption

The experiments show that as the size of the file increases, time consumed to encrypt the file also increases. But, the time for key generation remains almost constant irrespective of the file size.

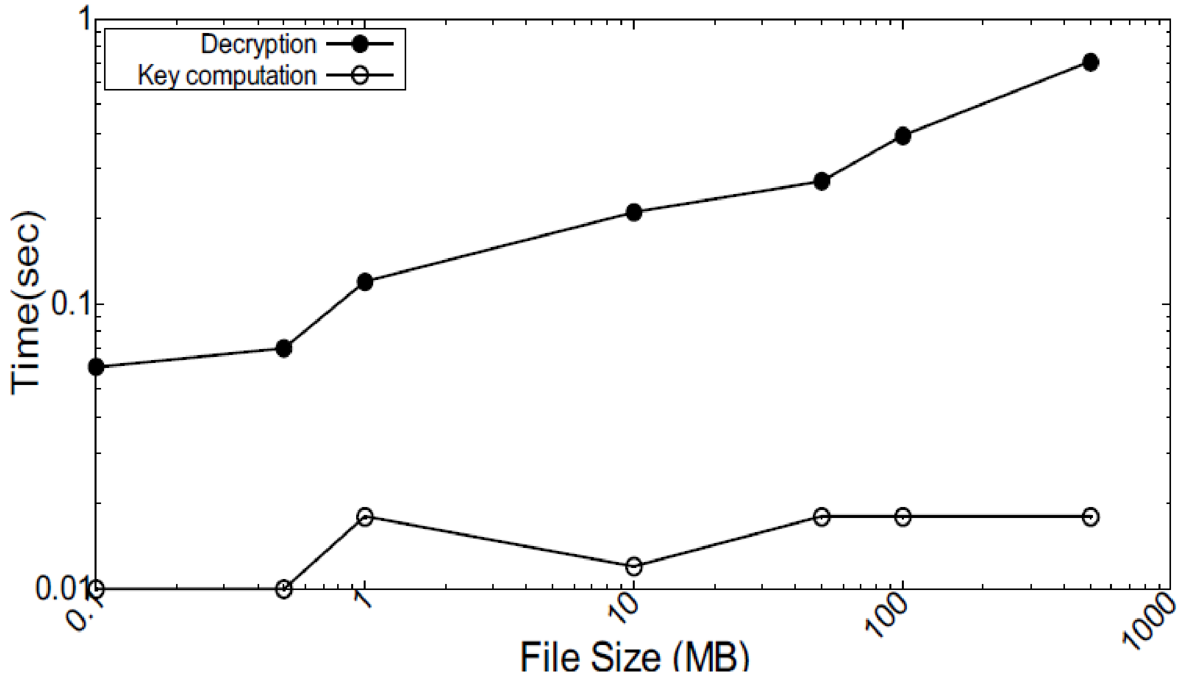

Figure 11. Time consumption for file decryption

The time required for decryption has been shown in Figure 11. A similar pattern has been observed for decryption too. The time required for decryption also shows an increasing pattern as the file size increases.

5.2.3. File Upload/Download

The file upload time here refers to the total time that is taken from the point where the request is sent to CS. It includes the following –

1. Key generation
2. Time required for file encryption/decryption
3. Time for file upload/download
4. Time taken by other processes performed during file upload/download

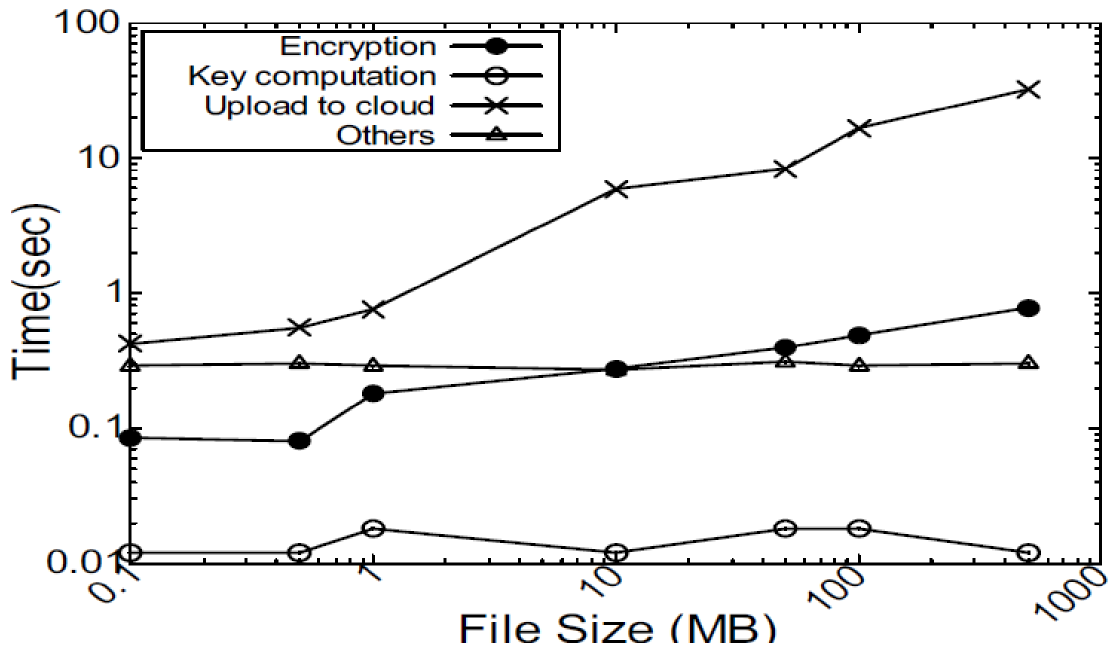

Figure 12. Time consumption for file upload

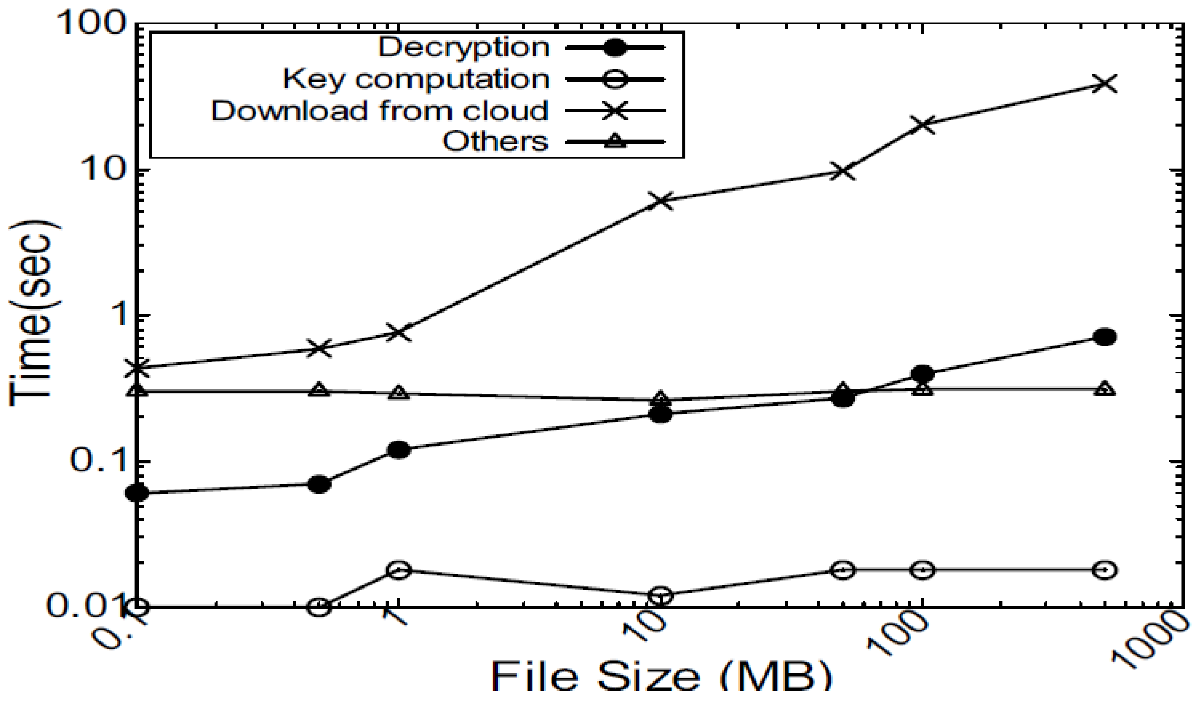

Figure 13. Time consumption for file download

The results in Figure 12 and 13 show that the total time taken to either upload or download a file increases as the size of the file increases. With the increase in file size, encryption/decryption increases but, at the same time, the upload/download time also increases. Since, the time consumed for key generation remains constant, it is independent of file size. As the size of file increases, the effect of time taken for key generation on the total time for file upload considerably reduces. This reveals that the scheme does not incur any high computation overhead.

Table 4. Comparison of key generation time (in seconds)

COMPARISON OF KEY GENERATION TIME				
No. of users	[5]	[2]	[1]	SeDaSC
10	1.494	1.594	1.534	0.004
20	1.598	1.741	1.606	0.00425
30	1.673	2.321	1.684	0.00476
40	1.791	1.888	1.799	0.005
50	1.907	1.952	1.866	0.00512
60	1.954	2.193	1.923	0.0055
70	1.994	2.286	2.034	0.00598
80	2.092	2.694	2.129	0.00632
90	2.401	2.827	2.388	0.00664
100	2.495	2.887	2.545	0.00697

Table 5. Comparison of turnaround time (in seconds)

Comparison of turnaround time								
FS (MB)	[5]		[2]		[1]		SeDaSC	
	UL	DL	UL	DL	UL	DL	UL	DL
0.1	0.9	0.81	1.4	0.99	1.48	1.15	0.8	0.8
0.5	1.18	0.96	1.48	1.03	1.89	1.31	0.94	0.96
1	1.8	1.39	2.06	1.48	2.9	1.85	1.24	1.18
10	13.05	9.91	14.95	9.9	14.59	10.45	6.43	6.48
50	53.68	33.45	58.56	35.57	60.37	35.9	9.01	10.24
100	99.69	57.14	112.41	59.14	155.15	61.59	17.37	20.68
500	369.72	215.3	492.03	229.81	872.09	400.21	33.24	39.25

Captions for the table are following.

FS = File Size, UL = Upload, DL = Download.

The SeDaSC methodology was compared with the schemes presented in [5], [2] and [1]. This comparison was based on the time consumption during key generation when the group is created and the turnaround time for encryption and decryption. The comparison of key generation time is provided in Table 4. Table 5 shows the turnaround time for encryption and decryption. Both the tables reveal that the SeDaSC methodology outperforms the other techniques due to absence of heavy computations.

6. CONCLUSION

In a nutshell, the proposed methodology SeDaSC focuses on sharing data in the cloud in a secure manner. This methodology provides secure data storage, data confidentiality along with backward and forward access control and at the same time low computation overhead as there is no re-encryption. It secures data from both the cloud service provider and unauthorized users as data is encrypted by a trusted party before being uploaded to the cloud. This methodology uses a single symmetric key for encryption/decryption. This key is split into two key shares for each user. One of the key share stays with the trusted party, while the other key share is sent to the corresponding user. The key is deleted after encryption/decryption operation using secure overwriting to avoid any threats from the users or the cloud. This methodology was formally verified using the High Level Petri Nets (HLPN), SMT-Lib, and Z3 solver. It was also implemented in Visual Studio to evaluate its performance which revealed that SeDaSC can be practically used for secure data sharing in cloud.

REFERENCES

[1]. Khan, A. N., Kiah, M. L. M., Madani, S. A., Ali, M., Khan, A. ur R., & Shamshirband, S. (2014). Incremental proxy re-encryption scheme for mobile cloud computing environment. *The Journal of Supercomputing*, *68*(2), 624–651.

[2]. Seo, S.-H., Nabeel, M., Ding, X., & Bertino, E. (2014). An Efficient Certificateless Encryption for Secure Data Sharing in Public Clouds. *IEEE Transactions on Knowledge and Data Engineering*, *26*(9), 2107–2119.

[3]. Thilakanathan, D., Chen, S., Nepal, S., & Calvo, R. A. (2014). Secure Data Sharing in the Cloud. In S. Nepal & M. Pathan (Eds.), *Security, Privacy and Trust in Cloud Systems* (pp. 45–72). Berlin, Heidelberg: Springer Berlin Heidelberg.

[4]. Tysowski, P. K., & Hasan, M. A. (2011). Re-Encryption-Based Key Management Towards Secure and Scalable Mobile Applications in Clouds. *IACR Cryptology ePrint Archive*, *2011*, 668.

[5]. Xu, L., Wu, X., & Zhang, X. (2012). CL-PRE: a certificateless proxy re-encryption scheme for secure data sharing with public cloud. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security* (pp. 87–88).

[6]. Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE, 77*(4), 541-580.

[7]. De Moura, L., & Bjørner, N. (2009). Satisfiability modulo theories: An appetizer. In *Formal Methods: Foundations and Applications* (pp. 23–36). Springer.

[8]. Malik, S. U. R., Srinivasan, S. K., Khan, S. U., & Wang, L. (2012). A methodology for OSPF routing protocol verification. In *Proceedings of international conference on scalable computing and communications (ScalCom), Changzhou, China*.

[9]. Yu, S., Wang, C., Ren, K., & Lou, W. (2010). Achieving secure, scalable and fine-grained data access control in cloud computing. In *INFOCOM, 2010 Proceedings IEEE* (pp. 1–9).

[10]. Barker, E., & Roginsky, A. (2011). Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. *NIST Special Publication*, *800*, 131A.

[11]. Lenstra, A. K., & Verheul, E. R. (2001). Selecting cryptographic key sizes. *Journal of Cryptology*, *14*(4), 255–293.

[12]. Chen, Y.-R., & Tzeng, W.-G. (2012). Efficient and Provably-Secure Group Key Management Scheme Using Key Derivation (pp. 295–302). IEEE.