SeDaSC: Secure Data Sharing in Clouds

Mazhar Ali, *Student Member, IEEE*, Revathi Dhamotharan, Eraj Khan, Samee U. Khan, *Senior Member, IEEE*, Athanasios V. Vasilakos, *Senior Member, IEEE*, Keqin Li, *Fellow, IEEE*, and Albert Y. Zomaya, *Fellow, IEEE*

***Abstract*—Cloud storage is an application of clouds that liberates organizations from establishing in-house data storage systems. However, cloud storage gives rise to security concerns. In case of group-shared data, the data face both cloud-specific and conventional insider threats. Secure data sharing among a group that counters insider threats of legitimate yet malicious users is an important research issue. In this paper, we propose the Secure Data Sharing in Clouds (SeDaSC) methodology that provides: 1) data confidentiality and integrity; 2) access control; 3) data sharing (forwarding) without using compute-intensive reencryption; 4) insider threat security; and 5) forward and backward access control. The SeDaSC methodology encrypts a file with a single encryption key. Two different key shares for each of the users are generated, with the user only getting one share. The possession of a single share of a key allows the SeDaSC methodology to counter the insider threats. The other key share is stored by a trusted third party, which is called the cryptographic server. The SeDaSC methodology is applicable to conventional and mobile cloud computing environments. We implement a working prototype of the SeDaSC methodology and evaluate its performance based on the time consumed during various operations. We formally verify the working of SeDaSC by using high-level Petri nets, the Satisfiability Modulo Theories Library, and a Z3 solver. The results proved to be encouraging and show that SeDaSC has the potential to be effectively used for secure data sharing in the cloud.**

***Index Terms*—Access control, cloud computing, high-level Petri nets (HLPNs), modeling, Satisfiability Modulo Theory (SMT), Scyther, verification.**

I. Introduction

CLOUD computing is rapidly emerging due to the provisioning of elastic, flexible, and on-demand storage and computing services for customers [1]. Organizations with a low budget can now utilize high computing and storage services without heavily investing in infrastructure and maintenance [2], [3]. However, the loss of control over data and computation raises many security concerns for organizations, thwarting the wide adaptability of the public cloud. The loss of control over data and the storage platform also motivates cloud customers to maintain the access control over data (individual data and the data shared among a group of users through the public cloud) [4]. Moreover, the privacy and confidentiality of the data is also recommended to be cared for by the customers [5]. The confidentiality management by a customer ensures that the cloud does not learn any information about the customer data. Cryptography is used as a typical tool to provide confidentiality and privacy services to the data [5]. The data are usually encrypted before storing to the cloud. The access control, key management, encryption, and decryption processes are handled by the customers to ensure data security [6]. However, when the data are to be shared among a group, the cryptographic services need to be flexible enough to handle different users, exercise the access control, and manage the keys in an effective manner to safeguard data confidentiality [7]. The data handling among a group has certain additional characteristics as opposed to two-party communication or the data handling belonging to a single user. The existing, departing, and newly joining group members can prove to be an insider threat violating data confidentiality and privacy [7]. Insider threats can prove to be more devastating due to the fact that they are generally launched by trusted entities. Due to the fact that people trust insider entities, the research community focuses more on outsider attackers. Nevertheless, multiple security issues can arise due to different users in a group. We discuss some of the issues in the following discussion.

A single key shared between all group members will result in the access of past data to a newly joining member. The aforesaid situation violates the confidentiality and the principle of least privilege [8]. Likewise, a departing member can access future communication. Therefore, in group-shared data, the inside members might generate the issue of backward access control (a new user accessing past data) and forward access control (a departing user accessing future data) [8]. The simple solution of rekeying (generating a new key, decrypting all the data, and reencrypting with the new key) does not prove to be scalable for frequent changes in the group membership [7].

A separate key for every user is a cumbersome solution. The data must be separately encrypted for every user in such a scenario. The changes in the data require the decryption of all of the copies of the users and encryption again with the modified contents [7].

The existing and legitimate group members might show illegitimate behavior to manipulate the data [3]. The presence of the entire symmetric key with a user allows a malicious user

Manuscript received September 24, 2014; revised November 13, 2014; accepted November 30, 2014.

M. Ali, R. Dhamotharan, and S. U. Khan are with the Department of Electrical and Computer Engineering, College of Engineering, North Dakota State University, Fargo, ND 58108-6050 USA (e-mail: mazhar.ali@ndsu.edu; revathi.dhamotharan@ndsu.edu; samee.khan@ndsu.edu).

E. Khan is with the Department of Computer Science, COMSATS Institute of Information Technology, 22060 Abbottabad, Pakistan (e-mail: eraj@ciit.net.pk).

A. V. Vasilakos is with the Department of Computer Science, College of Computer Science and Engineering, Kuwait University, 13060 Safat, Kuwait (e-mail: vasilako@cs.ku.edu.kw).

K. Li is with the Department of Computer Science, School of Science and Engineering, State University of New York, New Paltz, NY 12561-2443 USA (e-mail: lik@newpaltz.edu).

A. Y. Zomaya is with the School of Information Technologies, The University of Sydney, Sydney, NSW 2006 Australia (e-mail: albert.zomaya@sydney.edu.au).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSYST.2014.2379646

to turn into an insider threat [3]. The data can be decrypted, modified, and reencrypted by a malicious insider within a group. Consequently, a legitimate user in the group may access certain unauthorized files within the group [9]. On the other hand, it is necessary for a user to possess a key to conduct various operations on the data. The possession of the key also implicitly proves the legitimacy of a user to operate on the data [9]. Nevertheless, simultaneously dealing with both the issues related to the key is an important issue that needs to be addressed effectively.

In this paper, we propose a methodology named Secure Data Sharing in Clouds (SeDaSC) that deals with the aforementioned security requirements of shared group data within the cloud. The SeDaSC methodology works with three entities as follows: 1) users; 2) a cryptographic server (CS); and 3) the cloud. The data owner submits the data, the list of the users, and the parameters required for generating an access control list (ACL) to the CS. The CS is a trusted third party and is responsible for key management, encryption, decryption, and access control. The CS generates the symmetric key and encrypts the data with the generated key. Subsequently, for each user in the group, the CS divides the key into two parts such that a single part alone cannot regenerate the key. Successively, the original key is deleted through secure overwriting [10]. One part of the key is transmitted to the corresponding user in the group, whereas the other part is maintained by the CS within the ACL related to the data file. The ACL is generated through the parameter submitted by the data owner. The encrypted data are subsequently uploaded to the cloud for storage on behalf of the user. The user who wishes to access the data sends a download request to the CS. The CS, after authenticating the requesting user, receives the portion of the key from the user and subsequently downloads the data file from the cloud. The key is regenerated by operating on the user portion of the key, and the corresponding CS maintained portion for that particular user. The data are decrypted and sent back to the user. For a newly joining member, the two portions of the key are generated, and the user is added to the ACL. For a departing member, the record is deleted from the ACL. The departing member cannot decrypt the data on its own as he/she only possesses a portion of the key. Similarly, no frequent decryption and reencryption are needed in case of changes in the group membership. Moreover, SeDaSC can be used with the mobile cloud computing paradigm in addition to conventional cloud computing due to the fact that compute-intensive operations are performed by the CS. The working of the SeDaSC methodology is shown in Fig. 1, and the details are provided in Section III. Our major contributions, as reported in this paper, are as follows.

- The proposed methodology ensures the confidentiality of the data on the cloud by using symmetric encryption.
- The secure data sharing over the cloud among the group of users is ensured without the elliptic curve or bilinear Diffie–Hellman problem (BDH) cryptographic reencryption.
- The possession of a portion of the key secures the data against malicious insiders within the group.
- The proposed SeDaSC methodology secures the data against issues of forward and backward access control that arise due to insider threats.

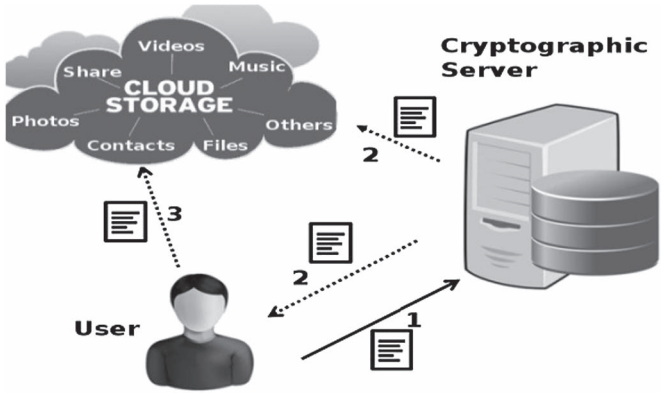

Fig. 1. Basic idea for the SeDaSC methodology.

- We perform formal modeling and verification of the SeDaSC methodology by using high-level Petri nets (HLPNs), the Satisfiability Modulo Theory Library (SMT-Lib), and a Z3 solver.

The remainder of this paper is organized as follows. Section II provides the overview of the related work in the field. In Section III, we present the details of the proposed SeDaSC methodology. Section IV provides the discussion on the services provided by the proposed methodology. Formal modeling and verification of SeDaSC are detailed in Section V. Section VI explains the experimental results, and Section VII concludes this paper.

II. Related Work

Xu *et al.* [9] proposed a certificateless proxy reencryption (CL-PRE) scheme for securely sharing the data within a group in the public cloud. In the CL-PRE scheme, the data owner encrypts the data with the symmetric key. Subsequently, the symmetric key is encrypted with the public key of the data owner. Both the encrypted data and the key are uploaded to the cloud. The encrypted key is reencrypted by the cloud (that acts as a proxy reencryption agent) that becomes decryptable by the user's private key. The public–private keys generated in the proposed scheme are not based on the certificates. The user's identity is used to generate the public–private key pair. The proxy reencryption is based on bilinear pairing and the BDH that makes the CL-PRE scheme computationally intensive. The computational cost of the bilinear pairing is high as compared with the standard operations in finite fields.

To reduce the computational overhead of bilinear pairing, Seo *et al.* [11] introduced a mediated certificateless encryption approach for data sharing in the public cloud that avoids bilinear pairing. In the proposed scheme, the cloud generates the public–private key pairs for all of the users and transmits the public keys to all of the participating users. Partial decryption is performed at the cloud. Due to the fact that key management and partial decryption are handled by the cloud, user revocation is easier to handle. However, the proposed scheme treats the public cloud both as a trusted and untrusted entity at the same time. From a security perspective, it is not recommended to shift the key generation process to the shared multitenant public cloud environment. Moreover, the decryption is performed twice in the system that reduces the advantage of not pairing to some extent.

Khan *et al.* [7] also utilized the El-Gamal cryptosystem and bilinear pairing for the sharing of sensitive information in the cloud. Moreover, the proposed scheme in [7] utilized the concept of incremental cryptography that divides the data into blocks and incrementally encrypts the blocks. The proposed scheme uses a trusted third party as a proxy that performs the compute-intensive operations of key generation, reencryption, and managing access to the data. However, the computational complexities of bilinear pairing still exist in the system.

Chen and Tzeng [8] proposed a methodology based on the shared key derivation method for securing data sharing among a group. The methodology uses a binary tree for the computation of keys. However, the computational cost of the proposed scheme is high as the rekeying mechanism is heavily employed in the proposed scheme. Moreover, the scheme is not tailored for public cloud systems because certain operations require centralized mediations. A similar Rivest–Shamir–Adleman (RSA)-based approach was also proposed in [12]. However, the scheme was vulnerable against collusion attacks.

The SeDaSC methodology, which is proposed in this paper, securely shares the data among a group without using the El-Gamal cryptosystem, the BDH, and bilinear pairing. The SeDaSC methodology is based on symmetric cryptography without reencryption. The aforesaid properties avoid computationally intensive operations and make the SeDaSC methodology a lightweight methodology. Moreover, the forward and backward access control is ensured by only allowing user access to a portion of the key that prohibits insiders to launch individual or coordinated attacks on the data.

III. SeDaSC

In this section, we present the design of our proposed methodology SeDaSC that secures the sharing and forwarding of data among a group without involving reencryption in the cloud environment.

A. Entities

The SeDaSC methodology has the following entities.

Cloud: The cloud provides storage services to the user. The data on the cloud need to be secured against privacy breaches. The confidentiality of the data is ensured by storing encrypted data over the cloud. The cloud in the SeDaSC methodology only involves basic cloud operations of file upload and download. Therefore, no changes at the protocol or implementation level on the cloud are required.

CS: The CS is a trusted party and is responsible for security operations, such as key management, encryption, decryption, the management of the ACL for providing confidentiality, and secure data forwarding among the group. The users of SeDaSC are required to be registered with the CS to obtain the security services. The CS is assumed to be a secure entity in the proposed methodology. The CS can be maintained by an organization or can be owned by a third-party provider. However, the CS maintained by an organization will generate more trust in the system.

Users: The users are the clients of the storage cloud. For each data file, one user will be the owner of the file, whereas the others in the group will be the data consumers. The owner of the file decides the access rights of the other group members. The access rights are granted and revoked based on the decision of the owner. The access rights are managed by the CS in the form of an ACL file. A separate ACL is maintained for each of the data files.

B. Cryptographic Keys

The SeDaSC methodology maintains a single cryptographic key for each of the data files. However, after encryption/decryption, the whole key is not stored and possessed by any of the involved parties. The key is partitioned into two constituent parts and are possessed by different entities. The following are the keys that are used within SeDaSC.

Symmetric Key K: K is a random secret generated by the CS for each of the data files. The length of K in SeDaSC is 256 bits, as is recommended by most of the standards regarding key length for symmetric key algorithms (SKAs). However, the length of the key can be altered according to the requirements of the underlying SKA. K is obtained in a two-step process. In the first step, a random number R of length 256 bits is generated such that $R = \{0, 1\}^{256}$. In the next step, R is passed through a hash function that could be any hash function with a 256-bit output. In our case, we used secure hash algorithm 256 (SHA-256). The second step completely randomizes the initial user-derived random number R. The output of the hash function is termed as K and is used in symmetric key encryption [e.g., the Advanced Encryption Standard (AES)] for securing the data.

CS Key Share K_i: For each of the users in the group, the CS generates K_i, such that $K_i = \{0, 1\}^{256}$. K_i serves as the CS portion of the key and is used to compute K whenever an encryption/decryption request is received by the CS. Moreover, it is ensured by comparison that the distinct K_i is generated for every file user.

User Key Share K_i': K_i' is computed for each of the users in the group as follows:

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

Algorithm 1 Key Generation and Encryption

Input:
F, the ACL, the SKA, the 256-bit hash function H_f
Compute:
$R = \{0, 1\}^{256}$
$K = H_f(R)$
$C = \text{SKA}(F, K)$
for each user i in **the ACL, do**
$K_i = \{0, 1\}^{256}$
$K_i = K \oplus K_i$
Add K_i' for user i in the ACL
Send K_i' for user i
end for
delete (K)
delete (K_i')
return C to the owner or upload to the cloud.

K_i' serves as the user portion of the key and is used to compute K when needed.

C. SeDaSC Design

In this section, we present the design of SeDaSC. In particular, we propose several cryptographic key operations that enable SeDaSC to achieve security goals.

1) File Upload: Whenever a need to share data among the group arises, the owner of the file sends the encryption request to the CS. The request is accompanied by the file (F) and a list (L) of users that are to be granted access to the file. L also contains the access rights for each of the users. The users may have READ-only and/or READ–WRITE access to the file. Other parameters can be also set to enforce fine-grained access control over the data. L is used to generate the ACL for the data by the CS. L is sent to the CS only if the data are to be shared with a new proposed group. If the group already exists, the encryption request will not contain L; rather, the group ID of the existing group will be sent. The CS, after receiving the encryption request for the file, generates the ACL from the list and creates a group of the users. The ACL is separately maintained for each file. The ACL contains information regarding the file such as its unique ID, size, owner ID, the list of the user IDs with whom the file is being shared, and other metadata. If the group already existed, only the ACL for the file is created. Next, the CS generates K according to the procedure defined in Section III-B and encrypts the file with an appropriate symmetric block cipher (we have used the AES for encryption purposes). The result is an encrypted file (C). Subsequently, the CS generates K_i and K_i' for every user and deletes K by secure overwriting. Secure overwriting is a concept in which the bits in the memory are constantly flipped to make sure that a memory cell never grips a charge for enough duration for it to be remembered and recovered. The K_i for each user is inserted into the ACL for later use. To protect the integrity of the file, the CS also computes the hash-based message authentication code (HMAC) signature on every encrypted file. A similar procedure for the HMAC key is adopted. However, the HMAC key is kept by the CS only. The encrypted data, the group ID (in the case of a newly generated group), and the K_i' for the owner are sent to the requesting data owner. The group ID and the K_i' for the rest of the group users are directly sent to them over a secure communication channel. The public keys of the group users can be also used to transmit the user portion of the key. We have used the public keys of the users to transmit the key portions. The user, after receiving C, uploads it to the cloud. K is deleted via secure overwriting from the CS after the encryption process. Fig. 2 shows the file upload operation. Algorithm 1 shows the key generation and encryption process at the CS. It is noteworthy that the key generation process is executed once when the group is initiated and the first file is submitted for encryption. Moreover, a newly joining member also activates the key generation but only for the new member.

It is important to note that, after the encryption of the data at the CS, the uploading of the file to the cloud can be handled in two possible ways. In the first option, the encrypted data can be sent to the user who uploads it to the cloud, as explained earlier in this section. In the second option, the CS can be delegated the authority to upload the file to the cloud on behalf of the user. We have used the second option in our implementation. The dashed line in Fig. 2 depicts the second option.

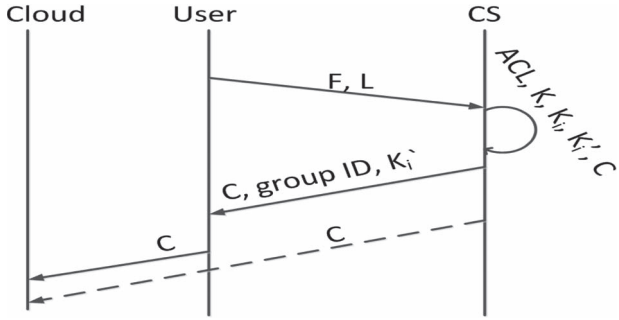

Fig. 2. File upload.

Algorithm 2 Decryption Algorithm

Input:
C, the ACL, the 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 the access denied message to the user
else
  $K = K_i \oplus K_i'$
  $F = \text{SKA}(C, K)$
  send F to the user
end if
delete (K)
delete (K_i').

2) File Download: The authorized user sends a download request to the CS or downloads the encrypted file (C) from the cloud and sends the decryption request to the CS. The cloud verifies the authorization of the user through a locally maintained ACL. The decryption request is accompanied by the user portion of the key, i.e., K_i', along with other authentication credentials. The CS computes K by applying XOR operation over K_i' and the corresponding K_i from the ACL. As each of the users correspond to a different pair of K_i and K_i', none of the users can use other users' K_i' to masquerade identity. Subsequently, the CS proceeds with the decryption process after verifying the integrity of the file. If the correct K_i' is received by the CS, the result will be a successful decryption process; otherwise, the decryption will fail. After successful decryption, the file is sent to the requesting user through a secure communication channel that could be Secure Sockets Layer (SSL) or Internet Protocol Security (IPSec) channels. K is deleted via secure overwriting from the CS after decryption. The users are authenticated before the request processing according to standard procedures. The process is highlighted in Fig. 3. Algorithm 2 presents the decryption process.

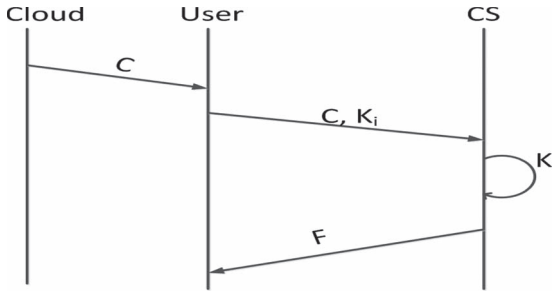

Fig. 3. File download.

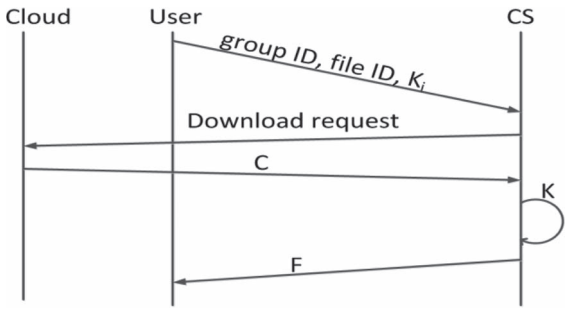

Fig. 4. File download: A special case.

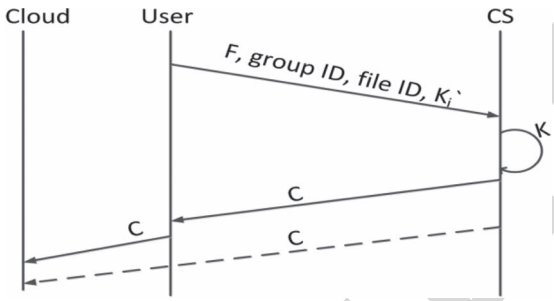

Fig. 5. File update.

Similar to the file upload process, the downloading of the file can be also done by the CS on behalf of the user. In the aforesaid case, the decryption request is sent to the CS along with the group ID, the file ID, and K_i'. The CS, after authenticating the user, sends the download request to the cloud for the specified file. The cloud sends the encrypted file (C) to the CS. The rest of the process for the decryption is the same. The download process in the second aforementioned case is shown in Fig. 4.

3) File Update: Updating the file has a similar procedure to that of uploading the file. The difference is that, while updating, all of the activities related to the creation of the ACL and key generation are not carried out. The user, who has downloaded the file and made any changes, sends an update request to the CS. The request contains the group ID, the file ID, and K_i', along with the file to be encrypted after changes. The CS verifies that the user has the WRITE access to the file from the corresponding ACL. In the case of a valid update request, the CS computes K by XORing K_i and K_i', encrypts the file, and performs the HMAC calculations. The encrypted file is sent to the user or uploaded to the cloud. K is deleted afterward. Fig. 5 shows the update process in the SeDaSC methodology.

4) New Group User Inclusion: If a new user joins the group, the addition of the user is made on the request of the file owner. The request contains the user ID of the joining user, along with the access control parameters to be included in the ACL, and the group ID. The parameters include the IDs of the files for which the user has been granted access rights. It also includes the details indicating the READ and/or WRITE rights granted to the user. Alternatively, the date can be mentioned from which the access rights are valid for the user. This ensures the backward access control for the joining member. The CS, after receiving the joining request, updates the ACLs related to the files for which the access is granted. The key shares are generated, and the user shares are sent to the user along with the corresponding file IDs.

5) Departing Group User: The CS is notified about a departing member by the group owner. The CS removes all of the records for the departing user from the ACLs of the related files. As the whole key is not possessed by the group members, the departing member (even being malicious) will be unable to decrypt any of the group data files. Even the presence of encrypted files with a malicious departing member will not affect the privacy of the data. The malicious member will be unable to construct the whole key for decryption. Therefore, the forward access control is also ensured by the SeDaSC methodology. The next section discusses how different security services are achieved by the SeDaSC methodology.

IV. Discussion on SeDaSC

The SeDaSC methodology 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 the aforementioned services are achieved.

We do not consider the cloud to be a secure and trustful entity in the context of SeDaSC. Multitenancy, virtualization, and a 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 the case of SeDaSC, the file is encrypted with K. K is generated at the CS and is deleted right after utilization. The CS or the user cannot reconstruct K alone. For confidentiality, the data cannot be leaked unless the attacker gains access to K. K in its entirety is not stored anywhere, and neither does it travel on the communication channel. Therefore, the access to K is a difficult task. Although an attacker gets hold of the user share, i.e., K_i', he/she will have to guess the other share correctly. The guess or random generation is to be made from a total of $2^{256} - 1$ possible shares. The probability of generating the correct share is $(1/(2^{256} - 1)) = 8.636 \times 10^{-78}$, 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 reencryption 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 CS key shares. After authentication, the user share of the key is used, along with the CS share, to generate K. As the user share is only possessed by a valid user, only a valid user can lead to successful encryption/decryption of the data.

The division and dispersal of the key also helps counter the insider malicious users within the group. The ACL is separately maintained 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 the malicious user. Furthermore, the absence of the entire key with the user and the ACL collectively ensures 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 [7]. The aforesaid schemes require the reencryption of the data each time the access to the data is requested by any user other than the owner. The El-Gamal cryptosystem is computationally intensive. Moreover, reencryption at each access adds to the overhead. The SeDaSC methodology utilizes symmetric encryption, and the access to multiple users is achieved through key management, as explained in the preceding section. Therefore, the overhead of the SeDaSC methodology is fairly less as compared with the traditional El-Gamal-based reencryption systems.

V. Formal Analysis

Before going into the details of the formal analysis of the proposed methodology, we provide a brief introduction to HLPNs, the SMT-Lib, and a Z3 solver for better understanding of the reader.

A. HLPNs

Petri nets are used for the graphical and mathematical representation of the system. Petri nets can model a range of systems, such as distributed, parallel, concurrent, nondeterministic, stochastic, or asynchronous systems [13]. We have used a variant of a conventional Petri net called an HLPN. An HLPN is a seven-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 $P \cap T = \varnothing$. 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. R defines the set of rules for transitions. L is a label on F, and M_0 represents the initial marking [13]. 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.

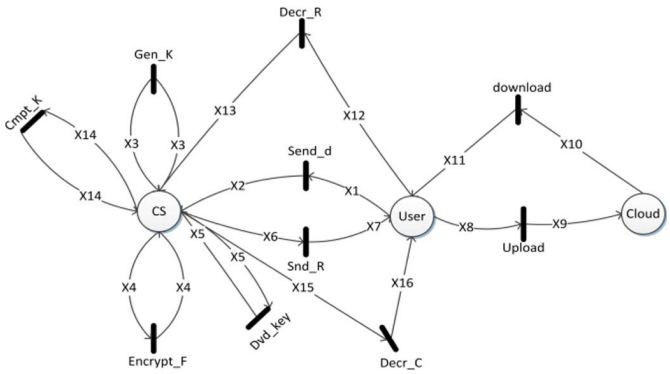

Fig. 6. HLPN model for SeDaSC.

TABLE I
Data Types for the HLPN Model

Types	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 type 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 II
Mapping of Data Types and Places

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

B. 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) [14]. We use a Z3 solver with the SMT-Lib that is not only theorem prover developed at Microsoft Research but is also an automated satisfiability checker. In addition, the Z3 solver determines whether the set of formulas are satisfiable in the built-in theories of the SMT-Lib. For the use of the SMT-Lib in the verification process, see [15].

The HLPN model for SeDaSC is shown in Fig. 6. The data types and mappings are shown in Tables I and II, respectively.

Whenever the data are to be shared among multiple users, the data owner sends the data file, i.e., F, to the CS. The list of the users is also sent along with F, along with other parameters discussed in Section III. The following rule is mapped to the transition Send_d of the HLPN:

$$R(\text{Send\_}d) = \forall x_1 \in X_1, \forall x_2 \in X_2 | x_2[1] := x_1[1] \wedge x_2[2] := x_1[2] \wedge X_2' = X_2 \cup \{x_2\}. \quad (2)$$

The CS generates a symmetric key, i.e., K, and other parameters according to the previously explained procedure. The following formula operates on transition Gen_K to depict the process:

$$R(\text{Gen\_}K) = \forall\, x\_3 \in X\_3 | x_3[3] = \text{gen\_grpID}(x_3[3]) \wedge x_3[4] := \text{gen}_{K(\cdot)} \wedge X'_3 = X_3 \cup \{x_3\}. \quad (3)$$

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

$$R(\text{Encrypt\_}F) = \forall\, x_4 \in X_4 | x_4[8] = \text{hash}\,(x_4[1]) \wedge x_4[7] := \text{encrypt}\,(x_4[1],\, x_4[4]) \wedge X'_4 = X_4 \cup \{x_4\}. \quad (4)$$

The CS computes the two constituent shares of K, i.e., K_i and K'_i, for each of the users in the ACL and deletes K afterward. Transition Dvd_K depicts the procedure with the following formula:

$$R(\text{Dvd\_}K) = \forall\, x_5 \in X_5 | x_5[5] := \text{gen\_}K_i(\,) \wedge x_5[6] := x_5[4] \oplus x_5[5] \wedge \text{over\_write}(x_5[4]) \wedge X'_5 = X_5 \cup \{x_5\}. \quad (5)$$

The encrypted data C, along with the hash value, the group ID, and K'_i, are sent to the data owner. The procedure is detailed in Section III. The following formula at transition Snd_R shows the following process:

$$\begin{aligned} R(\text{Snd\_}R) &= \forall\, x_6 \in X_6, \forall\, x_7 \in X_7 | x_7[3] \\ &:= x_6[6] \wedge x_7[4] := x_6[3] \wedge x_7[5] \\ &:= x_6[7] \wedge x_7[6] \\ &:= x_6[8] \wedge\ \text{over\_write}(x_6[6]) \wedge X'_6 \\ &= X_6 \cup \{x_6\} \wedge X'_7 = X_7 \cup \{x_7\}. \end{aligned} \quad (6)$$

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

$$R(\text{Upload}) = \forall\, x_8 \in X_8,\, \forall\, x_9 \in X_9 | x_9[1] = x_8[7] \wedge x_9[2] := x_8[8] \wedge X'_9 = X_9 \cup \{x_9\}. \quad (7)$$

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

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

The user sends a decryption request to the CS along with C, U_i, the group ID, and K'_i. The following rule maps to transition Decr_R:

$$\begin{aligned} R(\text{Decr\_}R) &= \forall\, x_{12} \in X_{12},\, \forall\, x_{13} \in X_{13} | x_{13}[2] \\ &:= x_{12}[2] \wedge x_{13}[3] = x_{12}[4] \wedge x_{13}[6] \\ &:= x_{12}[3] \wedge x_{13}[7] = x_{12}[5] \wedge X'_{13} \\ &= X_{13} \cup \{x_{13}\}. \end{aligned} \quad (9)$$

TABLE III
HARDWARE SPECIFICATIONS FOR CS AND USER CLIENT MACHINES

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

The CS, after verifying the authorization status of the user from the ACL, computes K according to the procedure defined in Section III. The following transition and rule shows the process:

$$R(\text{Cmpt\_}K) = \forall\, x_{14} \in X_{14} | x_{14}[4] = x_{14}[5] \oplus x_{14}[6] \wedge X'_{14} = X_{14} \cup \{x_{14}\}. \quad (10)$$

The CS decrypts the data and sends it back to the user. K and K'_i are deleted subsequently. Transition Decr_C shows the process as follows:

$$R(\text{Decr\_}C) = \forall\, x_{15} \in X_{15},\, \forall\, x_{16} \in X_{16} | x_{16}[1] := \text{decrypt}\,(x_{15}[7],\, x_{15}[4]) \wedge X'_{16} = X_{16} \cup \{x_{16}\}. \quad (11)$$

C. Verification of Properties

The properties that are verified are the following.

- 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.
- A valid user in the group leads to the generation of a valid K by contributing a valid K'_i.
- A malicious user outside the group, if somehow gets access to the encrypted file, cannot lead to its decryption.

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

VI. PERFORMANCE EVALUATION

A. Experimental Setup

To evaluate the performance of the proposed methodology, we implemented the SeDaSC methodology in Visual Studio 2010 C# using the .Net 4 framework. As discussed earlier, the proposed methodology consists of three entities, i.e., the cloud, the CS, and the users. The Amazon Web Services' software development kit and the .Net application programming interfaces were used to communicate with Amazon S3, which serves as the cloud server in our implementation. The CS is implemented as a third party. The functionality required by the user is implemented as a client application that connects with the CS to receive the services. The hardware characteristics for the CS and the user client are shown in Table III.

The communication between the entities was accomplished using .Net libraries (System.Net.Security and System.Net. Sockets). The classes TcpClient and TcpListener have been used

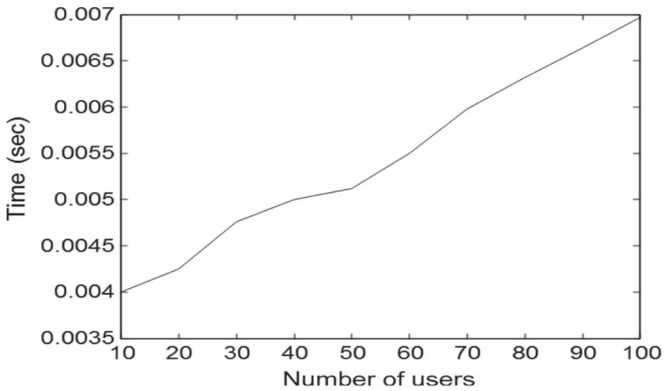

Fig. 7. Time consumption for key generation.

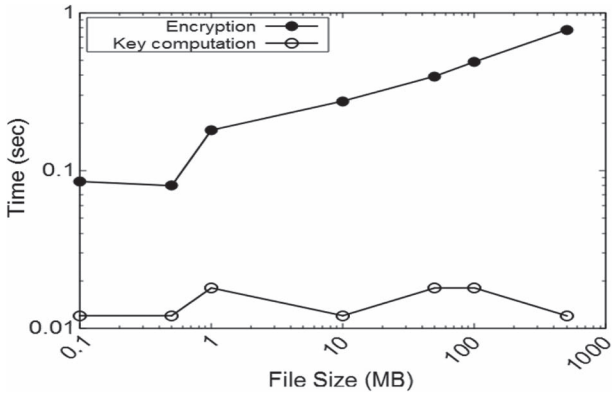

Fig. 8. Performance of file encryption for SeDaSC.

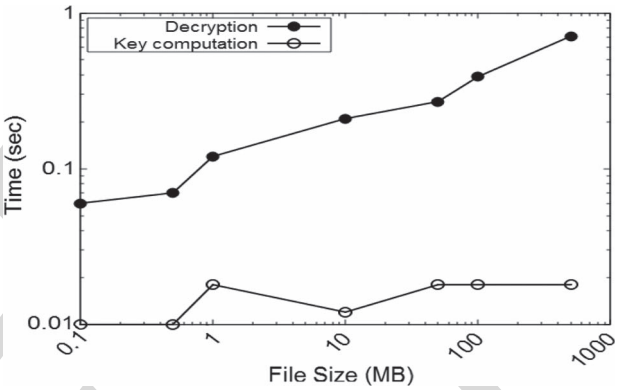

Fig. 9. Performance of file decryption for SeDaSC.

to implement the Transmission Control Protocol (TCP). The communication was then secured using the SSLStream class. The scheme uses the SHA-256 hash function for generating keys and the AES for encryption and decryption. The scheme was implemented using a .Net library, i.e., System.Security. Cryptography. The class SHA256CryptoServiceProvider within the library was used to access all of the methods related to SHA-256. All of the cryptographic operations, i.e., encryption and decryption, were implemented using the AES class that represents the base abstract class for the AES algorithm.

B. Results

The SeDaSC methodology has been evaluated for the following three different cases.

1) Key Generation: As described in Section III, there is only one symmetric key generated for each file. However, the key shares are separately computed for every user in the group. The shares are computed at the time of file submission. We evaluated SeDaSC for time consumption in key generation. The time is computed for different numbers of users. We set the number of users to be 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100. The results are shown in Fig. 7. In general, the time consumption for key generation increases with the increase in the number of users. However, it may be noted that the increase in the time consumption is not uniformly proportional to the increase in the number of users. For example, key generation takes 0.004 s for 10 users, and the time increases to 0.00512 s in the case of 50 users. The time has not increased in the same proportion as the number of users. Moreover, the jump in the time consumption varies as the number of users increases from 20 to 50. This may be attributed to the variation in the amount of time allotted to the application by the processor according to the processing situation of the system. Nevertheless, the time for key generation varies between 0.004 and 0.00697 s. The time for key generation is a slight overhead that is only generated once at the time of file submission in the group. A newly joining member will only consume the time for the generation of key shares that would be nominal as is computed for only a single user.

2) Encryption and Decryption: We evaluated the time consumption during the encryption and decryption of the file with varying file sizes. The file sizes used were 0.1, 0.5, 1, 10, 50, 100, and 500 MB. We have observed in Section III that the CS has to compute K before encryption and decryption. Therefore, the time to compute K is also compared with the total encryption and decryption times. The purpose is to observe the time overhead of the key computation over the total encryption and decryption times. The results for encryption and decryption are highlighted in Figs. 8 and 9, respectively.

Fig. 8 shows that, as expected, the time for encryption increases with the increase in the file size. However, the time for the computation of K almost remains constant with negligible change that may be due to the processing conditions at that point of time. This is because the time for the computation of K is independent of the file size. The comparative analysis shows that, with smaller file sizes, the percentage of the key computation time is high in comparison with the total encryption time. However, with the increase in the file size, the proportion of the key computation time in the total encryption time decreases rapidly. In the case of the 100-KB file, the key computation time constitutes 15% of the total encryption time. However, the increase in the file size (1 MB) drops the proportion to 10%. With further increase in the file size (10 MB), the percentage of the key computation time falls to 4.3%. The trends continue, and with a file size of 500 MB, the percentage remains merely at 1.54%. It is also noteworthy that the total key computation time ranges between 0.012 and 0.018 s.

Fig. 9 illustrates the results for decryption. The results show the similar trend for decryption, as was the case with encryption. The key computation time makes a high proportion of

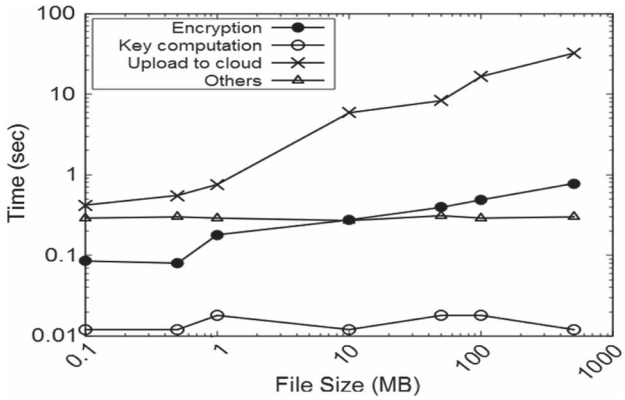

Fig. 10. Performance of file uploads for SeDaSC.

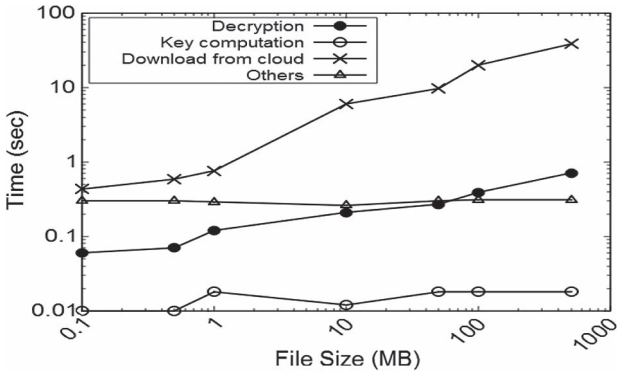

Fig. 11. Performance of file downloads for SeDaSC.

the total decryption time with small file sizes. However, with reasonably good file sizes, the key computation times makes a negligible proportion of the total consumed time. In the case of decryption, the percentage of the key computation ranges between 16.66% in the case of the 100-KB file and 2.53% for a file size of 500 MB.

3) File Upload/Download: We also evaluated the SeDaSC methodology on the basis of the total time consumed to upload/download a file to/from the cloud. The total time is composed of the time from the time of submission of request to the CS to the point of time at which the file is uploaded/downloaded to/from the cloud. The following times are included in the total time:

1) the key computation time;
2) the encryption/decryption time;
3) the upload/download time; and
4) the time of request and other related data submission to the CS and the cloud.

Fig. 10 shows the results for the upload time. All of the constituent times are represented by separate line graphs. The term "others" refers to the fourth constituent time discussed previously. In general, the time to upload the data increased with the increase in the file size. However, in some cases, the marginal increase in the file upload time was small that may be due to the network condition at various times. Nevertheless, the file upload time was dependent on the network conditions. Similar to the results in Section VI-B2, the key computation time remained almost constant and was independent of the file size. The encryption time increased with the increase in the file size. The other times almost remained constant and were also independent of the file size. It may be noted that the time for the key computation is negligible as compared with the total time consumed because it does not involve heavy computations.

Fig. 11 shows the results for the download operation involved in downloading the file from the cloud and the subsequent decryption process. The trend of results is similar as in the case of file upload. However, the times in decryption and download are changed.

We have compared the SeDaSC methodology with the schemes presented in [7], [9], and [11]. The comparison is based on the time consumption during key generation when the group is created and on the turnaround time for encryption and decryption. The comparison of key generation times is provided in Table IV. Table V shows the turnaround times for encryption and decryption. Both of these tables reveal that the SeDaSC methodology outperforms the other techniques due to the absence of heavy computations.

TABLE IV
COMPARISON OF KEY GENERATION TIMES

No. of users	[9]	[11]	[7]	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 V
COMPARISON OF TURNAROUND TIMES

FS	[9]		[11]		[7]		SeDaSC	
(MB)	UL	DL	UL	DL	UL	DL	UL	DL
0.1	0.90	0.81	1.4	0.99	1.48	1.15	0.80	0.80
0.5	1.18	0.96	1.48	1.03	1.89	1.31	0.94	0.96
1	1.80	1.39	2.06	1.48	2.90	1.85	1.24	1.18
10	13.05	9.91	14.95	9.90	14.59	10.45	6.43	6.48
50	53.68	33.45	58.56	35.57	60.37	35.90	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.

VII. CONCLUSION

We proposed the SeDaSC methodology, which is a cloud storage security scheme for group data. The proposed methodology provides data confidentiality, secure data sharing without reencryption, access control for malicious insiders, and forward and backward access control. Moreover, the SeDaSC

methodology provides assured deletion by deleting the parameters required to decrypt a file. The encryption and decryption functionalities are performed at the CS that is a trusted third party in the SeDaSC methodology. The proposed methodology can be also employed to mobile cloud computing due to the fact that compute-intensive tasks are performed at the CS. The working of SeDaSC was formally analyzed using HLPNs, the SMT-Lib, and a Z3 solver. The performance of the SeDaSC methodology was evaluated based on the time consumption during the key generation, file upload, and file download operations. The results revealed that the SeDaSC methodology can be practically used in the cloud for secure data sharing among the group.

In the future, the proposed methodology can be extended by limiting the trust level in the CS. This will further enhance the system to cope with insider threats. Moreover, the response of the methodology with varying key sizes can be evaluated.

REFERENCES

[1] A. Abbas and S. U. Khan, "A review on the State-of-the-art privacy preserving approaches in e-health clouds," *IEEE J. Biomed. Health Informat.*, vol. 18, no. 1, pp. 1431–1441, Jul. 2014.

[2] K. Alhamazani *et al.*, "An overview of the commercial cloud monitoring tools: Research dimensions, design issues, state-of-the-art," *Computing*, DOI: 10.1007/s00607-014-0398-5, 2014, to be published.

[3] A. N. Khan, M. L. M. Kiah, S. U. Khan, and S. A. Madani, "Towards secure mobile cloud computing: A survey," *Future Gen. Comput. Syst.*, vol. 29, no. 5, pp. 1278–1299, Jul. 2013.

[4] L. Wei, H. Zhu, Z. Cao, Y. Chen, and A. V. Vasilakos, "Security and privacy for storage and computation in cloud computing," *Inf. Sci.*, vol. 258, pp. 371–386, Feb. 2014.

[5] Cloud security Alliance, "Security guidelines for critical areas of focus in cloud computing v3.0," 2011.

[6] D. Chen *et al.*, "Fast and scalable multi-way analysis of massive neural data," *IEEE Trans. Comput.*, DOI: 10.1109/TC.2013.2295806, 2014, to be published.

[7] A. N. Khan, M. M. Kiah, S. A. Madani, M. Ali, and S. Shamshir-band, "Incremental proxy re-encryption scheme for mobile cloud computing environment," *J. Supercomput.*, vol. 68, no. 2, pp. 624–651, May 2014.

[8] Y. Chen and W. Tzeng, "Efficient and provably-secure group key management scheme using key derivation," in *Proc. IEEE 11th Int. Conf. TrustCom*, 2012, pp. 295–302.

[9] L. Xu, X. Wu, and X. Zhang, "CL-PRE: A certificateless proxy re-encryption scheme for secure data sharing with public cloud," in *Proc. 7th ACM Symp. Inf. , Comput. Commun. Security*, 2012, pp. 87–88.

[10] P. Gutmann, "Secure deletion of data from magnetic and solid-state memory," in *Proc. 6th USENIX Security Symp. Focusing Appl. Cryptography*, 1996, p. 8.

[11] S. Seo, M. Nabeel, X. Ding, and E. Bertino, "An Efficient Certificateless Encryption for Secure Data Sharing in Public Clouds," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 9, pp. 2107–2119, Sep. 2013.

[12] Y. Chen, J. D. Tygar, and W. Tzeng, "Secure group key management using uni-directional proxy re-encryption schemes," in *Proc. IEEE INFOCOM*, pp. 1952–1960.

[13] T. Murata, "Petri Nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[14] L. Moura and N. Bjrner, "Satisfiability modulo theories: An appetizer," in *Proc. Formal Methods, Found. Appl.*, vol. 5902, *Lecture Notes in Computer Science*, 2009, pp. 23–36.

[15] S. U. R. Malik, S. K. Srinivasan, S. U. Khan, and L. Wang, "A methodology for OSPF routing protocol verification," in *Proc. 12th Int. Conf. ScalCom*, Changzhou, China, Dec. 2012, pp. 1–5.

Mazhar Ali (S'14) is currently working toward the Ph.D. degree in the Department of Electrical and Computer Engineering, College of Engineering, North Dakota State University, Fargo, ND, USA.

His research interests include information security, formal verification, and cloud computing systems.

Revathi Dhamotharan is currently working toward the M.S. degree in electrical and computer engineering in the Department of Electrical and Computer Engineering, College of Engineering, North Dakota State University, Fargo, ND, USA.

Her research interests include cryptography and security.

Eraj Khan received the Ph.D. degree in communication security from Lancaster University, Lancaster, U.K.

He is currently with the Department of Computer Science, COMSATS Institute of Information Technology, Abbottabad, Pakistan. His main areas of research interests include code-based cryptography and security in cloud computing.

Samee U. Khan (S'02–M'07–SM'12) received the Ph.D. degree in computer science from University of Texas, Arlington, USA.

He is currently an Associate Professor of electrical and computer engineering with the Department of Electrical and Computer Engineering, College of Engineering, North Dakota State University, Fargo, ND, USA. His research interests include topics such as sustainable computing, social networking, and reliability.

Athanasios V. Vasilakos (M'00–SM'11) received the Ph.D. degree in computer engineering from the University of Patras, Patras, Greece.

He is currently a Professor with the Department of Computer Science, College of Computer Science and Engineering, Kuwait University, Safat, Kuwait. His research interests include robustness, security, computer networks, and distributed systems.

Keqin Li (M'90–SM'96–F'15) is a Distinguished Professor with the Department of Computer Science, School of Science and Engineering, State University of New York, New Paltz, NY, USA. His research interests mainly include the areas of design and analysis of algorithms, parallel and distributed computing, and computer networking.

Albert Y. Zomaya (F'04) is currently the Chair Professor of high-performance computing with the School of Information Technologies, The University of Sydney, Sydney, Australia.

Mr. Zomaya is a Fellow of The Institution of Engineering and Technology and of the American Association for the Advancement of Science.

SeDaSC: Secure Data Sharing in Clouds

Mazhar Ali, *Student Member, IEEE*, Revathi Dhamotharan, Eraj Khan, Samee U. Khan, *Senior Member, IEEE*, Athanasios V. Vasilakos, *Senior Member, IEEE*, Keqin Li, *Fellow, IEEE*, and Albert Y. Zomaya, *Fellow, IEEE*

***Abstract*—Cloud storage is an application of clouds that liberates organizations from establishing in-house data storage systems. However, cloud storage gives rise to security concerns. In case of group-shared data, the data face both cloud-specific and conventional insider threats. Secure data sharing among a group that counters insider threats of legitimate yet malicious users is an important research issue. In this paper, we propose the Secure Data Sharing in Clouds (SeDaSC) methodology that provides: 1) data confidentiality and integrity; 2) access control; 3) data sharing (forwarding) without using compute-intensive reencryption; 4) insider threat security; and 5) forward and backward access control. The SeDaSC methodology encrypts a file with a single encryption key. Two different key shares for each of the users are generated, with the user only getting one share. The possession of a single share of a key allows the SeDaSC methodology to counter the insider threats. The other key share is stored by a trusted third party, which is called the cryptographic server. The SeDaSC methodology is applicable to conventional and mobile cloud computing environments. We implement a working prototype of the SeDaSC methodology and evaluate its performance based on the time consumed during various operations. We formally verify the working of SeDaSC by using high-level Petri nets, the Satisfiability Modulo Theories Library, and a Z3 solver. The results proved to be encouraging and show that SeDaSC has the potential to be effectively used for secure data sharing in the cloud.**

***Index Terms*—Access control, cloud computing, high-level Petri nets (HLPNs), modeling, Satisfiability Modulo Theory (SMT), Scyther, verification.**

I. Introduction

CLOUD computing is rapidly emerging due to the provisioning of elastic, flexible, and on-demand storage and computing services for customers [1]. Organizations with a low budget can now utilize high computing and storage services without heavily investing in infrastructure and maintenance [2], [3]. However, the loss of control over data and computation raises many security concerns for organizations, thwarting the wide adaptability of the public cloud. The loss of control over data and the storage platform also motivates cloud customers to maintain the access control over data (individual data and the data shared among a group of users through the public cloud) [4]. Moreover, the privacy and confidentiality of the data is also recommended to be cared for by the customers [5]. The confidentiality management by a customer ensures that the cloud does not learn any information about the customer data. Cryptography is used as a typical tool to provide confidentiality and privacy services to the data [5]. The data are usually encrypted before storing to the cloud. The access control, key management, encryption, and decryption processes are handled by the customers to ensure data security [6]. However, when the data are to be shared among a group, the cryptographic services need to be flexible enough to handle different users, exercise the access control, and manage the keys in an effective manner to safeguard data confidentiality [7]. The data handling among a group has certain additional characteristics as opposed to two-party communication or the data handling belonging to a single user. The existing, departing, and newly joining group members can prove to be an insider threat violating data confidentiality and privacy [7]. Insider threats can prove to be more devastating due to the fact that they are generally launched by trusted entities. Due to the fact that people trust insider entities, the research community focuses more on outsider attackers. Nevertheless, multiple security issues can arise due to different users in a group. We discuss some of the issues in the following discussion.

A single key shared between all group members will result in the access of past data to a newly joining member. The aforesaid situation violates the confidentiality and the principle of least privilege [8]. Likewise, a departing member can access future communication. Therefore, in group-shared data, the inside members might generate the issue of backward access control (a new user accessing past data) and forward access control (a departing user accessing future data) [8]. The simple solution of rekeying (generating a new key, decrypting all the data, and reencrypting with the new key) does not prove to be scalable for frequent changes in the group membership [7].

A separate key for every user is a cumbersome solution. The data must be separately encrypted for every user in such a scenario. The changes in the data require the decryption of all of the copies of the users and encryption again with the modified contents [7].

The existing and legitimate group members might show illegitimate behavior to manipulate the data [3]. The presence of the entire symmetric key with a user allows a malicious user

Manuscript received September 24, 2014; revised November 13, 2014; accepted November 30, 2014.

M. Ali, R. Dhamotharan, and S. U. Khan are with the Department of Electrical and Computer Engineering, College of Engineering, North Dakota State University, Fargo, ND 58108-6050 USA (e-mail: mazhar.ali@ndsu.edu; revathi.dhamotharan@ndsu.edu; samee.khan@ndsu.edu).

E. Khan is with the Department of Computer Science, COMSATS Institute of Information Technology, 22060 Abbottabad, Pakistan (e-mail: eraj@ciit.net.pk).

A. V. Vasilakos is with the Department of Computer Science, College of Computer Science and Engineering, Kuwait University, 13060 Safat, Kuwait (e-mail: vasilako@cs.ku.edu.kw).

K. Li is with the Department of Computer Science, School of Science and Engineering, State University of New York, New Paltz, NY 12561-2443 USA (e-mail: lik@newpaltz.edu).

A. Y. Zomaya is with the School of Information Technologies, The University of Sydney, Sydney, NSW 2006 Australia (e-mail: albert.zomaya@sydney.edu.au).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSYST.2014.2379646

to turn into an insider threat [3]. The data can be decrypted, modified, and reencrypted by a malicious insider within a group. Consequently, a legitimate user in the group may access certain unauthorized files within the group [9]. On the other hand, it is necessary for a user to possess a key to conduct various operations on the data. The possession of the key also implicitly proves the legitimacy of a user to operate on the data [9]. Nevertheless, simultaneously dealing with both the issues related to the key is an important issue that needs to be addressed effectively.

In this paper, we propose a methodology named Secure Data Sharing in Clouds (SeDaSC) that deals with the aforementioned security requirements of shared group data within the cloud. The SeDaSC methodology works with three entities as follows: 1) users; 2) a cryptographic server (CS); and 3) the cloud. The data owner submits the data, the list of the users, and the parameters required for generating an access control list (ACL) to the CS. The CS is a trusted third party and is responsible for key management, encryption, decryption, and access control. The CS generates the symmetric key and encrypts the data with the generated key. Subsequently, for each user in the group, the CS divides the key into two parts such that a single part alone cannot regenerate the key. Successively, the original key is deleted through secure overwriting [10]. One part of the key is transmitted to the corresponding user in the group, whereas the other part is maintained by the CS within the ACL related to the data file. The ACL is generated through the parameter submitted by the data owner. The encrypted data are subsequently uploaded to the cloud for storage on behalf of the user. The user who wishes to access the data sends a download request to the CS. The CS, after authenticating the requesting user, receives the portion of the key from the user and subsequently downloads the data file from the cloud. The key is regenerated by operating on the user portion of the key, and the corresponding CS maintained portion for that particular user. The data are decrypted and sent back to the user. For a newly joining member, the two portions of the key are generated, and the user is added to the ACL. For a departing member, the record is deleted from the ACL. The departing member cannot decrypt the data on its own as he/she only possesses a portion of the key. Similarly, no frequent decryption and reencryption are needed in case of changes in the group membership. Moreover, SeDaSC can be used with the mobile cloud computing paradigm in addition to conventional cloud computing due to the fact that compute-intensive operations are performed by the CS. The working of the SeDaSC methodology is shown in Fig. 1, and the details are provided in Section III. Our major contributions, as reported in this paper, are as follows.

- The proposed methodology ensures the confidentiality of the data on the cloud by using symmetric encryption.
- The secure data sharing over the cloud among the group of users is ensured without the elliptic curve or bilinear Diffie–Hellman problem (BDH) cryptographic reencryption.
- The possession of a portion of the key secures the data against malicious insiders within the group.
- The proposed SeDaSC methodology secures the data against issues of forward and backward access control that arise due to insider threats.

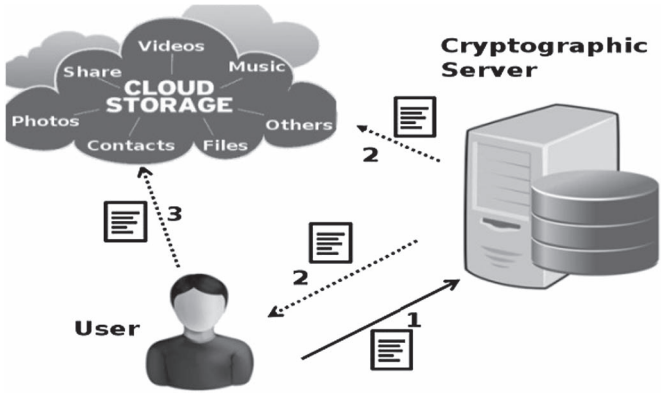

Fig. 1. Basic idea for the SeDaSC methodology.

- We perform formal modeling and verification of the SeDaSC methodology by using high-level Petri nets (HLPNs), the Satisfiability Modulo Theory Library (SMT-Lib), and a Z3 solver.

The remainder of this paper is organized as follows. Section II provides the overview of the related work in the field. In Section III, we present the details of the proposed SeDaSC methodology. Section IV provides the discussion on the services provided by the proposed methodology. Formal modeling and verification of SeDaSC are detailed in Section V. Section VI explains the experimental results, and Section VII concludes this paper.

II. Related Work

Xu *et al.* [9] proposed a certificateless proxy reencryption (CL-PRE) scheme for securely sharing the data within a group in the public cloud. In the CL-PRE scheme, the data owner encrypts the data with the symmetric key. Subsequently, the symmetric key is encrypted with the public key of the data owner. Both the encrypted data and the key are uploaded to the cloud. The encrypted key is reencrypted by the cloud (that acts as a proxy reencryption agent) that becomes decryptable by the user's private key. The public–private keys generated in the proposed scheme are not based on the certificates. The user's identity is used to generate the public–private key pair. The proxy reencryption is based on bilinear pairing and the BDH that makes the CL-PRE scheme computationally intensive. The computational cost of the bilinear pairing is high as compared with the standard operations in finite fields.

To reduce the computational overhead of bilinear pairing, Seo *et al.* [11] introduced a mediated certificateless encryption approach for data sharing in the public cloud that avoids bilinear pairing. In the proposed scheme, the cloud generates the public–private key pairs for all of the users and transmits the public keys to all of the participating users. Partial decryption is performed at the cloud. Due to the fact that key management and partial decryption are handled by the cloud, user revocation is easier to handle. However, the proposed scheme treats the public cloud both as a trusted and untrusted entity at the same time. From a security perspective, it is not recommended to shift the key generation process to the shared multitenant public cloud environment. Moreover, the decryption is performed twice in the system that reduces the advantage of not pairing to some extent.

Khan *et al.* [7] also utilized the El-Gamal cryptosystem and bilinear pairing for the sharing of sensitive information in the cloud. Moreover, the proposed scheme in [7] utilized the concept of incremental cryptography that divides the data into blocks and incrementally encrypts the blocks. The proposed scheme uses a trusted third party as a proxy that performs the compute-intensive operations of key generation, reencryption, and managing access to the data. However, the computational complexities of bilinear pairing still exist in the system.

Chen and Tzeng [8] proposed a methodology based on the shared key derivation method for securing data sharing among a group. The methodology uses a binary tree for the computation of keys. However, the computational cost of the proposed scheme is high as the rekeying mechanism is heavily employed in the proposed scheme. Moreover, the scheme is not tailored for public cloud systems because certain operations require centralized mediations. A similar Rivest–Shamir–Adleman (RSA)-based approach was also proposed in [12]. However, the scheme was vulnerable against collusion attacks.

The SeDaSC methodology, which is proposed in this paper, securely shares the data among a group without using the El-Gamal cryptosystem, the BDH, and bilinear pairing. The SeDaSC methodology is based on symmetric cryptography without reencryption. The aforesaid properties avoid computationally intensive operations and make the SeDaSC methodology a lightweight methodology. Moreover, the forward and backward access control is ensured by only allowing user access to a portion of the key that prohibits insiders to launch individual or coordinated attacks on the data.

III. SeDaSC

In this section, we present the design of our proposed methodology SeDaSC that secures the sharing and forwarding of data among a group without involving reencryption in the cloud environment.

A. Entities

The SeDaSC methodology has the following entities.

Cloud: The cloud provides storage services to the user. The data on the cloud need to be secured against privacy breaches. The confidentiality of the data is ensured by storing encrypted data over the cloud. The cloud in the SeDaSC methodology only involves basic cloud operations of file upload and download. Therefore, no changes at the protocol or implementation level on the cloud are required.

CS: The CS is a trusted party and is responsible for security operations, such as key management, encryption, decryption, the management of the ACL for providing confidentiality, and secure data forwarding among the group. The users of SeDaSC are required to be registered with the CS to obtain the security services. The CS is assumed to be a secure entity in the proposed methodology. The CS can be maintained by an organization or can be owned by a third-party provider. However, the CS maintained by an organization will generate more trust in the system.

Users: The users are the clients of the storage cloud. For each data file, one user will be the owner of the file, whereas the others in the group will be the data consumers. The owner of the file decides the access rights of the other group members. The access rights are granted and revoked based on the decision of the owner. The access rights are managed by the CS in the form of an ACL file. A separate ACL is maintained for each of the data files.

B. Cryptographic Keys

The SeDaSC methodology maintains a single cryptographic key for each of the data files. However, after encryption/decryption, the whole key is not stored and possessed by any of the involved parties. The key is partitioned into two constituent parts and are possessed by different entities. The following are the keys that are used within SeDaSC.

Symmetric Key K: K is a random secret generated by the CS for each of the data files. The length of K in SeDaSC is 256 bits, as is recommended by most of the standards regarding key length for symmetric key algorithms (SKAs). However, the length of the key can be altered according to the requirements of the underlying SKA. K is obtained in a two-step process. In the first step, a random number R of length 256 bits is generated such that $R = \{0, 1\}^{256}$. In the next step, R is passed through a hash function that could be any hash function with a 256-bit output. In our case, we used secure hash algorithm 256 (SHA-256). The second step completely randomizes the initial user-derived random number R. The output of the hash function is termed as K and is used in symmetric key encryption [e.g., the Advanced Encryption Standard (AES)] for securing the data.

CS Key Share K_i: For each of the users in the group, the CS generates K_i, such that $K_i = \{0, 1\}^{256}$. K_i serves as the CS portion of the key and is used to compute K whenever an encryption/decryption request is received by the CS. Moreover, it is ensured by comparison that the distinct K_i is generated for every file user.

User Key Share K_i': K_i' is computed for each of the users in the group as follows:

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

Algorithm 1 Key Generation and Encryption

Input:
F, the ACL, the SKA, the 256-bit hash function H_f
Compute:
$R = \{0, 1\}^{256}$
$K = H_f(R)$
$C = \text{SKA}(F, K)$
for each user i in **the ACL, do**
$K_i = \{0, 1\}^{256}$
$K_i = K \oplus K_i$
Add K_i' for user i in the ACL
Send K_i' for user i
end for
delete (K)
delete (K_i')
return C to the owner or upload to the cloud.

K_i' serves as the user portion of the key and is used to compute K when needed.

C. SeDaSC Design

In this section, we present the design of SeDaSC. In particular, we propose several cryptographic key operations that enable SeDaSC to achieve security goals.

1) File Upload: Whenever a need to share data among the group arises, the owner of the file sends the encryption request to the CS. The request is accompanied by the file (F) and a list (L) of users that are to be granted access to the file. L also contains the access rights for each of the users. The users may have READ-only and/or READ–WRITE access to the file. Other parameters can be also set to enforce fine-grained access control over the data. L is used to generate the ACL for the data by the CS. L is sent to the CS only if the data are to be shared with a new proposed group. If the group already exists, the encryption request will not contain L; rather, the group ID of the existing group will be sent. The CS, after receiving the encryption request for the file, generates the ACL from the list and creates a group of the users. The ACL is separately maintained for each file. The ACL contains information regarding the file such as its unique ID, size, owner ID, the list of the user IDs with whom the file is being shared, and other metadata. If the group already existed, only the ACL for the file is created. Next, the CS generates K according to the procedure defined in Section III-B and encrypts the file with an appropriate symmetric block cipher (we have used the AES for encryption purposes). The result is an encrypted file (C). Subsequently, the CS generates K_i and K_i' for every user and deletes K by secure overwriting. Secure overwriting is a concept in which the bits in the memory are constantly flipped to make sure that a memory cell never grips a charge for enough duration for it to be remembered and recovered. The K_i for each user is inserted into the ACL for later use. To protect the integrity of the file, the CS also computes the hash-based message authentication code (HMAC) signature on every encrypted file. A similar procedure for the HMAC key is adopted. However, the HMAC key is kept by the CS only. The encrypted data, the group ID (in the case of a newly generated group), and the K_i' for the owner are sent to the requesting data owner. The group ID and the K_i' for the rest of the group users are directly sent to them over a secure communication channel. The public keys of the group users can be also used to transmit the user portion of the key. We have used the public keys of the users to transmit the key portions. The user, after receiving C, uploads it to the cloud. K is deleted via secure overwriting from the CS after the encryption process. Fig. 2 shows the file upload operation. Algorithm 1 shows the key generation and encryption process at the CS. It is noteworthy that the key generation process is executed once when the group is initiated and the first file is submitted for encryption. Moreover, a newly joining member also activates the key generation but only for the new member.

It is important to note that, after the encryption of the data at the CS, the uploading of the file to the cloud can be handled in two possible ways. In the first option, the encrypted data can be sent to the user who uploads it to the cloud, as explained earlier in this section. In the second option, the CS can be delegated the authority to upload the file to the cloud on behalf of the user. We have used the second option in our implementation. The dashed line in Fig. 2 depicts the second option.

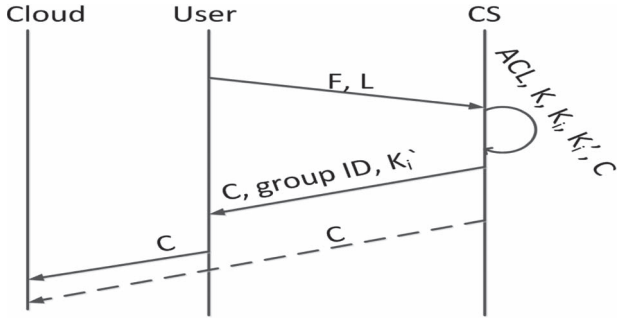

Fig. 2. File upload.

Algorithm 2 Decryption Algorithm

Input:
C, the ACL, the 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 the access denied message to the user
else
  $K = K_i \oplus K_i'$
  $F = \text{SKA}(C, K)$
  send F to the user
end if
delete (K)
delete (K_i').

2) File Download: The authorized user sends a download request to the CS or downloads the encrypted file (C) from the cloud and sends the decryption request to the CS. The cloud verifies the authorization of the user through a locally maintained ACL. The decryption request is accompanied by the user portion of the key, i.e., K_i', along with other authentication credentials. The CS computes K by applying XOR operation over K_i' and the corresponding K_i from the ACL. As each of the users correspond to a different pair of K_i and K_i', none of the users can use other users' K_i' to masquerade identity. Subsequently, the CS proceeds with the decryption process after verifying the integrity of the file. If the correct K_i' is received by the CS, the result will be a successful decryption process; otherwise, the decryption will fail. After successful decryption, the file is sent to the requesting user through a secure communication channel that could be Secure Sockets Layer (SSL) or Internet Protocol Security (IPSec) channels. K is deleted via secure overwriting from the CS after decryption. The users are authenticated before the request processing according to standard procedures. The process is highlighted in Fig. 3. Algorithm 2 presents the decryption process.

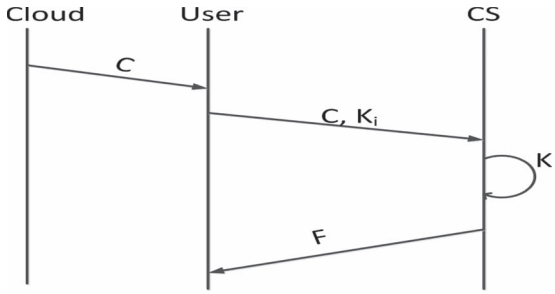

Fig. 3. File download.

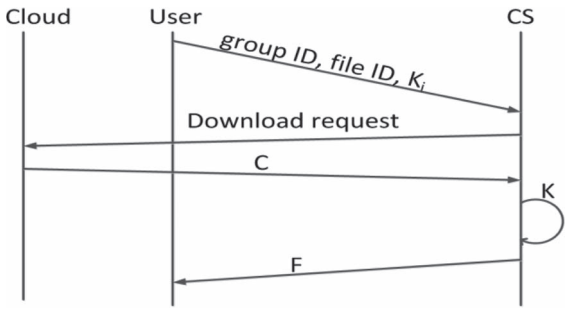

Fig. 4. File download: A special case.

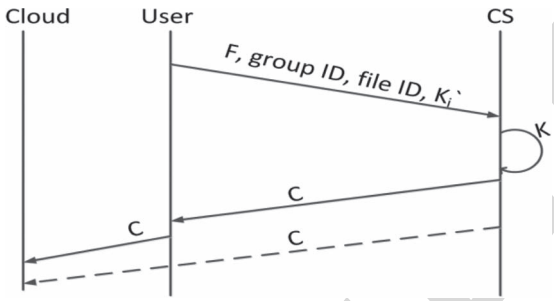

Fig. 5. File update.

Similar to the file upload process, the downloading of the file can be also done by the CS on behalf of the user. In the aforesaid case, the decryption request is sent to the CS along with the group ID, the file ID, and K_i'. The CS, after authenticating the user, sends the download request to the cloud for the specified file. The cloud sends the encrypted file (C) to the CS. The rest of the process for the decryption is the same. The download process in the second aforementioned case is shown in Fig. 4.

3) File Update: Updating the file has a similar procedure to that of uploading the file. The difference is that, while updating, all of the activities related to the creation of the ACL and key generation are not carried out. The user, who has downloaded the file and made any changes, sends an update request to the CS. The request contains the group ID, the file ID, and K_i', along with the file to be encrypted after changes. The CS verifies that the user has the WRITE access to the file from the corresponding ACL. In the case of a valid update request, the CS computes K by XORing K_i and K_i', encrypts the file, and performs the HMAC calculations. The encrypted file is sent to the user or uploaded to the cloud. K is deleted afterward. Fig. 5 shows the update process in the SeDaSC methodology.

4) New Group User Inclusion: If a new user joins the group, the addition of the user is made on the request of the file owner. The request contains the user ID of the joining user, along with the access control parameters to be included in the ACL, and the group ID. The parameters include the IDs of the files for which the user has been granted access rights. It also includes the details indicating the READ and/or WRITE rights granted to the user. Alternatively, the date can be mentioned from which the access rights are valid for the user. This ensures the backward access control for the joining member. The CS, after receiving the joining request, updates the ACLs related to the files for which the access is granted. The key shares are generated, and the user shares are sent to the user along with the corresponding file IDs.

5) Departing Group User: The CS is notified about a departing member by the group owner. The CS removes all of the records for the departing user from the ACLs of the related files. As the whole key is not possessed by the group members, the departing member (even being malicious) will be unable to decrypt any of the group data files. Even the presence of encrypted files with a malicious departing member will not affect the privacy of the data. The malicious member will be unable to construct the whole key for decryption. Therefore, the forward access control is also ensured by the SeDaSC methodology. The next section discusses how different security services are achieved by the SeDaSC methodology.

IV. Discussion on SeDaSC

The SeDaSC methodology 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 the aforementioned services are achieved.

We do not consider the cloud to be a secure and trustful entity in the context of SeDaSC. Multitenancy, virtualization, and a 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 the case of SeDaSC, the file is encrypted with K. K is generated at the CS and is deleted right after utilization. The CS or the user cannot reconstruct K alone. For confidentiality, the data cannot be leaked unless the attacker gains access to K. K in its entirety is not stored anywhere, and neither does it travel on the communication channel. Therefore, the access to K is a difficult task. Although an attacker gets hold of the user share, i.e., K_i', he/she will have to guess the other share correctly. The guess or random generation is to be made from a total of $2^{256} - 1$ possible shares. The probability of generating the correct share is $(1/(2^{256} - 1)) = 8.636 \times 10^{-78}$, 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 reencryption 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 CS key shares. After authentication, the user share of the key is used, along with the CS share, to generate K. As the user share is only possessed by a valid user, only a valid user can lead to successful encryption/decryption of the data.

The division and dispersal of the key also helps counter the insider malicious users within the group. The ACL is separately maintained 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 the malicious user. Furthermore, the absence of the entire key with the user and the ACL collectively ensures 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 [7]. The aforesaid schemes require the reencryption of the data each time the access to the data is requested by any user other than the owner. The El-Gamal cryptosystem is computationally intensive. Moreover, reencryption at each access adds to the overhead. The SeDaSC methodology utilizes symmetric encryption, and the access to multiple users is achieved through key management, as explained in the preceding section. Therefore, the overhead of the SeDaSC methodology is fairly less as compared with the traditional El-Gamal-based reencryption systems.

V. Formal Analysis

Before going into the details of the formal analysis of the proposed methodology, we provide a brief introduction to HLPNs, the SMT-Lib, and a Z3 solver for better understanding of the reader.

A. HLPNs

Petri nets are used for the graphical and mathematical representation of the system. Petri nets can model a range of systems, such as distributed, parallel, concurrent, nondeterministic, stochastic, or asynchronous systems [13]. We have used a variant of a conventional Petri net called an HLPN. An HLPN is a seven-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 $P \cap T = \varnothing$. 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. R defines the set of rules for transitions. L is a label on F, and M_0 represents the initial marking [13]. 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.

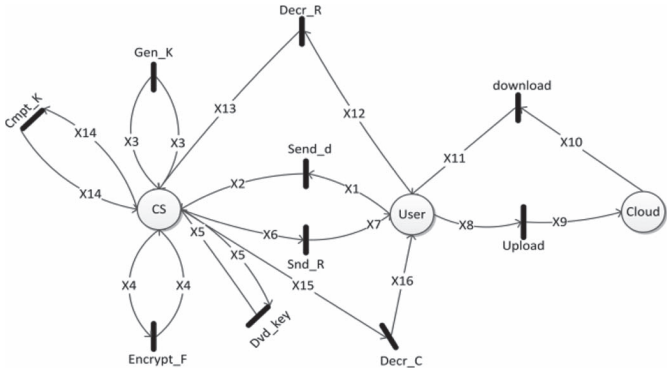

Fig. 6. HLPN model for SeDaSC.

TABLE I
Data Types for the HLPN Model

Types	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 type 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 II
Mapping of Data Types and Places

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

B. 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) [14]. We use a Z3 solver with the SMT-Lib that is not only theorem prover developed at Microsoft Research but is also an automated satisfiability checker. In addition, the Z3 solver determines whether the set of formulas are satisfiable in the built-in theories of the SMT-Lib. For the use of the SMT-Lib in the verification process, see [15].

The HLPN model for SeDaSC is shown in Fig. 6. The data types and mappings are shown in Tables I and II, respectively.

Whenever the data are to be shared among multiple users, the data owner sends the data file, i.e., F, to the CS. The list of the users is also sent along with F, along with other parameters discussed in Section III. The following rule is mapped to the transition Send_d of the HLPN:

$$R(\text{Send\_}d) = \forall x_1 \in X_1, \forall x_2 \in X_2 | x_2[1] := x_1[1] \wedge x_2[2] := x_1[2] \wedge X_2' = X_2 \cup \{x_2\}. \quad (2)$$

The CS generates a symmetric key, i.e., K, and other parameters according to the previously explained procedure. The following formula operates on transition Gen_K to depict the process:

$$R(\text{Gen\_}K) = \forall\, x\_3 \in X\_3 | x_3[3] = \text{gen\_grpID}(x_3[3]) \wedge x_3[4] := \text{gen}_{K(\cdot)} \wedge X'_3 = X_3 \cup \{x_3\}. \quad (3)$$

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

$$R(\text{Encrypt\_}F) = \forall\, x_4 \in X_4 | x_4[8] = \text{hash}\,(x_4[1]) \wedge x_4[7] := \text{encrypt}\,(x_4[1],\, x_4[4]) \wedge X'_4 = X_4 \cup \{x_4\}. \quad (4)$$

The CS computes the two constituent shares of K, i.e., K_i and K'_i, for each of the users in the ACL and deletes K afterward. Transition Dvd_K depicts the procedure with the following formula:

$$R(\text{Dvd\_}K) = \forall\, x_5 \in X_5 | x_5[5] := \text{gen\_}K_i(\,) \wedge x_5[6] := x_5[4] \oplus x_5[5] \wedge \text{over\_write}(x_5[4]) \wedge X'_5 = X_5 \cup \{x_5\}. \quad (5)$$

The encrypted data C, along with the hash value, the group ID, and K'_i, are sent to the data owner. The procedure is detailed in Section III. The following formula at transition Snd_R shows the following process:

$$\begin{aligned} R(\text{Snd\_}R) &= \forall\, x_6 \in X_6, \forall\, x_7 \in X_7 | x_7[3] \\ &:= x_6[6] \wedge x_7[4] := x_6[3] \wedge x_7[5] \\ &:= x_6[7] \wedge x_7[6] \\ &:= x_6[8] \wedge\ \text{over\_write}(x_6[6]) \wedge X'_6 \\ &= X_6 \cup \{x_6\} \wedge X'_7 = X_7 \cup \{x_7\}. \end{aligned} \quad (6)$$

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

$$R(\text{Upload}) = \forall\, x_8 \in X_8,\, \forall\, x_9 \in X_9 | x_9[1] = x_8[7] \wedge x_9[2] := x_8[8] \wedge X'_9 = X_9 \cup \{x_9\}. \quad (7)$$

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

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

The user sends a decryption request to the CS along with C, U_i, the group ID, and K'_i. The following rule maps to transition Decr_R:

$$\begin{aligned} R(\text{Decr\_}R) &= \forall\, x_{12} \in X_{12},\, \forall\, x_{13} \in X_{13} | x_{13}[2] \\ &:= x_{12}[2] \wedge x_{13}[3] = x_{12}[4] \wedge x_{13}[6] \\ &:= x_{12}[3] \wedge x_{13}[7] = x_{12}[5] \wedge X'_{13} \\ &= X_{13} \cup \{x_{13}\}. \end{aligned} \quad (9)$$

TABLE III
HARDWARE SPECIFICATIONS FOR CS AND USER CLIENT MACHINES

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

The CS, after verifying the authorization status of the user from the ACL, computes K according to the procedure defined in Section III. The following transition and rule shows the process:

$$R(\text{Cmpt\_}K) = \forall\, x_{14} \in X_{14} | x_{14}[4] = x_{14}[5] \oplus x_{14}[6] \wedge X'_{14} = X_{14} \cup \{x_{14}\}. \quad (10)$$

The CS decrypts the data and sends it back to the user. K and K'_i are deleted subsequently. Transition Decr_C shows the process as follows:

$$R(\text{Decr\_}C) = \forall\, x_{15} \in X_{15},\, \forall\, x_{16} \in X_{16} | x_{16}[1] := \text{decrypt}\,(x_{15}[7],\, x_{15}[4]) \wedge X'_{16} = X_{16} \cup \{x_{16}\}. \quad (11)$$

C. Verification of Properties

The properties that are verified are the following.

- 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.
- A valid user in the group leads to the generation of a valid K by contributing a valid K'_i.
- A malicious user outside the group, if somehow gets access to the encrypted file, cannot lead to its decryption.

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

VI. PERFORMANCE EVALUATION

A. Experimental Setup

To evaluate the performance of the proposed methodology, we implemented the SeDaSC methodology in Visual Studio 2010 C# using the .Net 4 framework. As discussed earlier, the proposed methodology consists of three entities, i.e., the cloud, the CS, and the users. The Amazon Web Services' software development kit and the .Net application programming interfaces were used to communicate with Amazon S3, which serves as the cloud server in our implementation. The CS is implemented as a third party. The functionality required by the user is implemented as a client application that connects with the CS to receive the services. The hardware characteristics for the CS and the user client are shown in Table III.

The communication between the entities was accomplished using .Net libraries (System.Net.Security and System.Net. Sockets). The classes TcpClient and TcpListener have been used

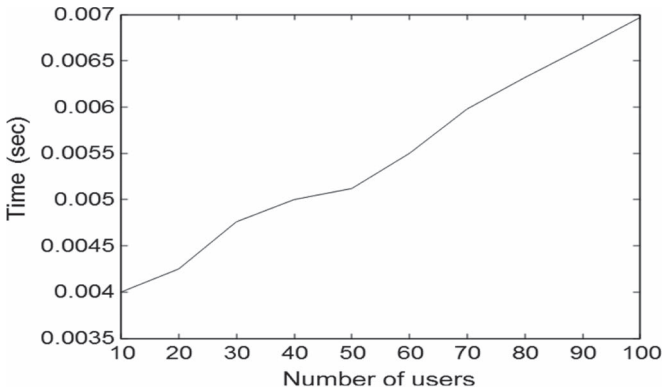

Fig. 7. Time consumption for key generation.

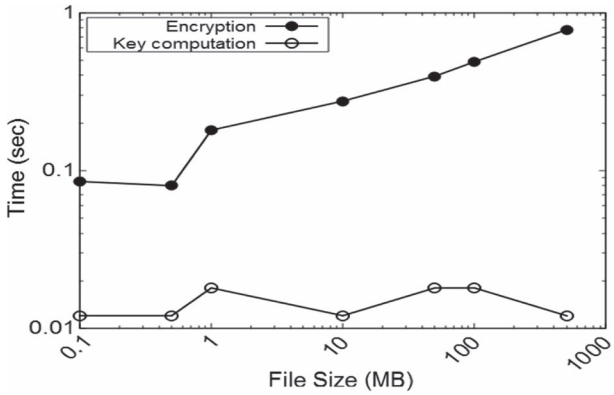

Fig. 8. Performance of file encryption for SeDaSC.

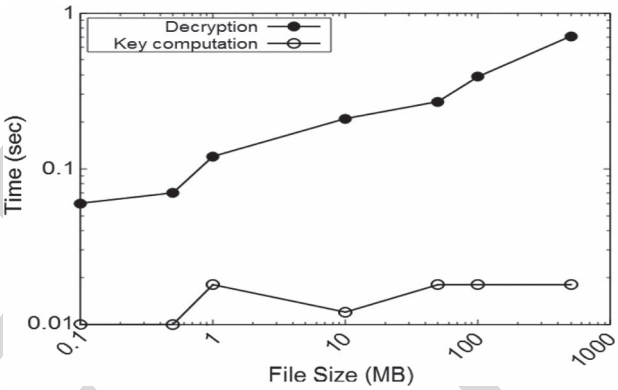

Fig. 9. Performance of file decryption for SeDaSC.

to implement the Transmission Control Protocol (TCP). The communication was then secured using the SSLStream class. The scheme uses the SHA-256 hash function for generating keys and the AES for encryption and decryption. The scheme was implemented using a .Net library, i.e., System.Security. Cryptography. The class SHA256CryptoServiceProvider within the library was used to access all of the methods related to SHA-256. All of the cryptographic operations, i.e., encryption and decryption, were implemented using the AES class that represents the base abstract class for the AES algorithm.

B. Results

The SeDaSC methodology has been evaluated for the following three different cases.

1) Key Generation: As described in Section III, there is only one symmetric key generated for each file. However, the key shares are separately computed for every user in the group. The shares are computed at the time of file submission. We evaluated SeDaSC for time consumption in key generation. The time is computed for different numbers of users. We set the number of users to be 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100. The results are shown in Fig. 7. In general, the time consumption for key generation increases with the increase in the number of users. However, it may be noted that the increase in the time consumption is not uniformly proportional to the increase in the number of users. For example, key generation takes 0.004 s for 10 users, and the time increases to 0.00512 s in the case of 50 users. The time has not increased in the same proportion as the number of users. Moreover, the jump in the time consumption varies as the number of users increases from 20 to 50. This may be attributed to the variation in the amount of time allotted to the application by the processor according to the processing situation of the system. Nevertheless, the time for key generation varies between 0.004 and 0.00697 s. The time for key generation is a slight overhead that is only generated once at the time of file submission in the group. A newly joining member will only consume the time for the generation of key shares that would be nominal as is computed for only a single user.

2) Encryption and Decryption: We evaluated the time consumption during the encryption and decryption of the file with varying file sizes. The file sizes used were 0.1, 0.5, 1, 10, 50, 100, and 500 MB. We have observed in Section III that the CS has to compute K before encryption and decryption. Therefore, the time to compute K is also compared with the total encryption and decryption times. The purpose is to observe the time overhead of the key computation over the total encryption and decryption times. The results for encryption and decryption are highlighted in Figs. 8 and 9, respectively.

Fig. 8 shows that, as expected, the time for encryption increases with the increase in the file size. However, the time for the computation of K almost remains constant with negligible change that may be due to the processing conditions at that point of time. This is because the time for the computation of K is independent of the file size. The comparative analysis shows that, with smaller file sizes, the percentage of the key computation time is high in comparison with the total encryption time. However, with the increase in the file size, the proportion of the key computation time in the total encryption time decreases rapidly. In the case of the 100-KB file, the key computation time constitutes 15% of the total encryption time. However, the increase in the file size (1 MB) drops the proportion to 10%. With further increase in the file size (10 MB), the percentage of the key computation time falls to 4.3%. The trends continue, and with a file size of 500 MB, the percentage remains merely at 1.54%. It is also noteworthy that the total key computation time ranges between 0.012 and 0.018 s.

Fig. 9 illustrates the results for decryption. The results show the similar trend for decryption, as was the case with encryption. The key computation time makes a high proportion of

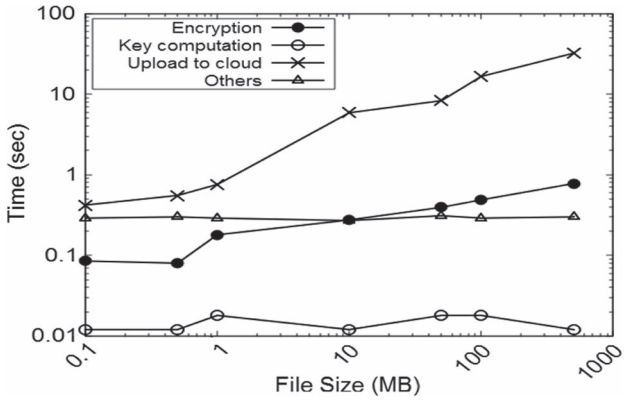

Fig. 10. Performance of file uploads for SeDaSC.

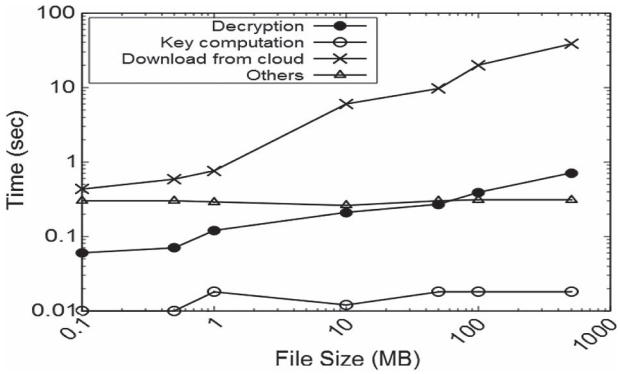

Fig. 11. Performance of file downloads for SeDaSC.

the total decryption time with small file sizes. However, with reasonably good file sizes, the key computation times makes a negligible proportion of the total consumed time. In the case of decryption, the percentage of the key computation ranges between 16.66% in the case of the 100-KB file and 2.53% for a file size of 500 MB.

3) File Upload/Download: We also evaluated the SeDaSC methodology on the basis of the total time consumed to upload/download a file to/from the cloud. The total time is composed of the time from the time of submission of request to the CS to the point of time at which the file is uploaded/downloaded to/from the cloud. The following times are included in the total time:

1) the key computation time;
2) the encryption/decryption time;
3) the upload/download time; and
4) the time of request and other related data submission to the CS and the cloud.

Fig. 10 shows the results for the upload time. All of the constituent times are represented by separate line graphs. The term "others" refers to the fourth constituent time discussed previously. In general, the time to upload the data increased with the increase in the file size. However, in some cases, the marginal increase in the file upload time was small that may be due to the network condition at various times. Nevertheless, the file upload time was dependent on the network conditions. Similar to the results in Section VI-B2, the key computation time remained almost constant and was independent of the file size. The encryption time increased with the increase in the file size. The other times almost remained constant and were also independent of the file size. It may be noted that the time for the key computation is negligible as compared with the total time consumed because it does not involve heavy computations.

Fig. 11 shows the results for the download operation involved in downloading the file from the cloud and the subsequent decryption process. The trend of results is similar as in the case of file upload. However, the times in decryption and download are changed.

We have compared the SeDaSC methodology with the schemes presented in [7], [9], and [11]. The comparison is based on the time consumption during key generation when the group is created and on the turnaround time for encryption and decryption. The comparison of key generation times is provided in Table IV. Table V shows the turnaround times for encryption and decryption. Both of these tables reveal that the SeDaSC methodology outperforms the other techniques due to the absence of heavy computations.

TABLE IV
COMPARISON OF KEY GENERATION TIMES

No. of users	[9]	[11]	[7]	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 V
COMPARISON OF TURNAROUND TIMES

FS	[9]		[11]		[7]		SeDaSC	
(MB)	UL	DL	UL	DL	UL	DL	UL	DL
0.1	0.90	0.81	1.4	0.99	1.48	1.15	0.80	0.80
0.5	1.18	0.96	1.48	1.03	1.89	1.31	0.94	0.96
1	1.80	1.39	2.06	1.48	2.90	1.85	1.24	1.18
10	13.05	9.91	14.95	9.90	14.59	10.45	6.43	6.48
50	53.68	33.45	58.56	35.57	60.37	35.90	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.

VII. CONCLUSION

We proposed the SeDaSC methodology, which is a cloud storage security scheme for group data. The proposed methodology provides data confidentiality, secure data sharing without reencryption, access control for malicious insiders, and forward and backward access control. Moreover, the SeDaSC

methodology provides assured deletion by deleting the parameters required to decrypt a file. The encryption and decryption functionalities are performed at the CS that is a trusted third party in the SeDaSC methodology. The proposed methodology can be also employed to mobile cloud computing due to the fact that compute-intensive tasks are performed at the CS. The working of SeDaSC was formally analyzed using HLPNs, the SMT-Lib, and a Z3 solver. The performance of the SeDaSC methodology was evaluated based on the time consumption during the key generation, file upload, and file download operations. The results revealed that the SeDaSC methodology can be practically used in the cloud for secure data sharing among the group.

In the future, the proposed methodology can be extended by limiting the trust level in the CS. This will further enhance the system to cope with insider threats. Moreover, the response of the methodology with varying key sizes can be evaluated.

REFERENCES

[1] A. Abbas and S. U. Khan, "A review on the State-of-the-art privacy preserving approaches in e-health clouds," *IEEE J. Biomed. Health Informat.*, vol. 18, no. 1, pp. 1431–1441, Jul. 2014.

[2] K. Alhamazani *et al.*, "An overview of the commercial cloud monitoring tools: Research dimensions, design issues, state-of-the-art," *Computing*, DOI: 10.1007/s00607-014-0398-5, 2014, to be published.

[3] A. N. Khan, M. L. M. Kiah, S. U. Khan, and S. A. Madani, "Towards secure mobile cloud computing: A survey," *Future Gen. Comput. Syst.*, vol. 29, no. 5, pp. 1278–1299, Jul. 2013.

[4] L. Wei, H. Zhu, Z. Cao, Y. Chen, and A. V. Vasilakos, "Security and privacy for storage and computation in cloud computing," *Inf. Sci.*, vol. 258, pp. 371–386, Feb. 2014.

[5] Cloud security Alliance, "Security guidelines for critical areas of focus in cloud computing v3.0," 2011.

[6] D. Chen *et al.*, "Fast and scalable multi-way analysis of massive neural data," *IEEE Trans. Comput.*, DOI: 10.1109/TC.2013.2295806, 2014, to be published.

[7] A. N. Khan, M. M. Kiah, S. A. Madani, M. Ali, and S. Shamshir-band, "Incremental proxy re-encryption scheme for mobile cloud computing environment," *J. Supercomput.*, vol. 68, no. 2, pp. 624–651, May 2014.

[8] Y. Chen and W. Tzeng, "Efficient and provably-secure group key management scheme using key derivation," in *Proc. IEEE 11th Int. Conf. TrustCom*, 2012, pp. 295–302.

[9] L. Xu, X. Wu, and X. Zhang, "CL-PRE: A certificateless proxy re-encryption scheme for secure data sharing with public cloud," in *Proc. 7th ACM Symp. Inf. , Comput. Commun. Security*, 2012, pp. 87–88.

[10] P. Gutmann, "Secure deletion of data from magnetic and solid-state memory," in *Proc. 6th USENIX Security Symp. Focusing Appl. Cryptography*, 1996, p. 8.

[11] S. Seo, M. Nabeel, X. Ding, and E. Bertino, "An Efficient Certificateless Encryption for Secure Data Sharing in Public Clouds," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 9, pp. 2107–2119, Sep. 2013.

[12] Y. Chen, J. D. Tygar, and W. Tzeng, "Secure group key management using uni-directional proxy re-encryption schemes," in *Proc. IEEE INFOCOM*, pp. 1952–1960.

[13] T. Murata, "Petri Nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[14] L. Moura and N. Bjrner, "Satisfiability modulo theories: An appetizer," in *Proc. Formal Methods, Found. Appl.*, vol. 5902, *Lecture Notes in Computer Science*, 2009, pp. 23–36.

[15] S. U. R. Malik, S. K. Srinivasan, S. U. Khan, and L. Wang, "A methodology for OSPF routing protocol verification," in *Proc. 12th Int. Conf. ScalCom*, Changzhou, China, Dec. 2012, pp. 1–5.

Mazhar Ali (S'14) is currently working toward the Ph.D. degree in the Department of Electrical and Computer Engineering, College of Engineering, North Dakota State University, Fargo, ND, USA.

His research interests include information security, formal verification, and cloud computing systems.

Revathi Dhamotharan is currently working toward the M.S. degree in electrical and computer engineering in the Department of Electrical and Computer Engineering, College of Engineering, North Dakota State University, Fargo, ND, USA.

Her research interests include cryptography and security.

Eraj Khan received the Ph.D. degree in communication security from Lancaster University, Lancaster, U.K.

He is currently with the Department of Computer Science, COMSATS Institute of Information Technology, Abbottabad, Pakistan. His main areas of research interests include code-based cryptography and security in cloud computing.

Samee U. Khan (S'02–M'07–SM'12) received the Ph.D. degree in computer science from University of Texas, Arlington, USA.

He is currently an Associate Professor of electrical and computer engineering with the Department of Electrical and Computer Engineering, College of Engineering, North Dakota State University, Fargo, ND, USA. His research interests include topics such as sustainable computing, social networking, and reliability.

Athanasios V. Vasilakos (M'00–SM'11) received the Ph.D. degree in computer engineering from the University of Patras, Patras, Greece.

He is currently a Professor with the Department of Computer Science, College of Computer Science and Engineering, Kuwait University, Safat, Kuwait. His research interests include robustness, security, computer networks, and distributed systems.

Keqin Li (M'90–SM'96–F'15) is a Distinguished Professor with the Department of Computer Science, School of Science and Engineering, State University of New York, New Paltz, NY, USA. His research interests mainly include the areas of design and analysis of algorithms, parallel and distributed computing, and computer networking.

Albert Y. Zomaya (F'04) is currently the Chair Professor of high-performance computing with the School of Information Technologies, The University of Sydney, Sydney, Australia.

Mr. Zomaya is a Fellow of The Institution of Engineering and Technology and of the American Association for the Advancement of Science.