Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/jides

A brief overview and an experimental evaluation of data confidentiality measures on the cloud

Hussain Aljafer[a], *Zaki Malik*[a,*], *Mohammed Alodib*[b], *Abdelmounaam Rezgui*[c]

[a] *Wayne State University, Detroit, MI 48202, USA*
[b] *Qassim University, Buraidah Al Qassim, Saudi Arabia*
[c] *New Mexico Tech., Socorro, NM 87801, USA*

ARTICLE INFO

Article history:
Received 23 October 2014
Received in revised form
19 January 2015
Accepted 15 February 2015
Published online 3 March 2015

Keywords:
Security and protection
Cloud
Encryption
Measurement

ABSTRACT

Due to the many advantages offered by the cloud computing paradigm, it is fast becoming an enabling technology for many organizations, and even individual users. Flexibility and availability are two of the most important features that promote the wide spread adoption of this technology. In cloud-based data storage scenarios, where the data is controlled by a third party (i.e. the cloud service provider), the data owner usually does not have full control of its data at all stages. Consequently, this poses a prime security threat, and a major challenge is the development of a secure protocol for data storage, sharing, and retrieval. In recent years, a number of research works have targeted this problem. In this paper, we discuss some of the major approaches for secure data sharing in the cloud computing environment. The goal is to provide a concise survey of existing solutions, discuss their benefits, and point out any shortcomings for future research. Specifically, we focus on the use of encryption schemes, and provide a comparative study of the major schemes, through implementation of some representative frameworks.

1. Introduction

In recent years, interest in cloud computing has gained considerable momentum. Cloud computing is centered on the notion of "services", which are independently developed and deployed artifacts to which clients subscribe on per need basis, and pay the service providers based on their usage. A service can formally be defined as a set of related functionalities that can be programmatically accessed and manipulated over the Web using a set of (XML-based) standards. Many organizations are using cloud based systems to store their data and important information. Apple's "iCloud" is a prime example where each user is granted a space on the cloud to store his/her data and retrieve it whenever, and where ever (i.e. using a number of devices in potentially different geographical locations) needed. Data security in this

Peer review under responsibility of Qassim University.
* *Corresponding author*. Tel.: +1 313 577 4987.
E-mail address: zaki@wayne.edu (Z. Malik).

http://dx.doi.org/10.1016/j.jides.2015.02.001

new paradigm is a major concern, as users and organizations want their data to be secured not only on the cloud but also when uploading the data, retrieving the data when the data is used by some Web services to answer the queries. In this regard, a major threat is when a third party may monitor the connections between the services or between the user and the services to obtain access to confidential data. However, threats do not only come from a third party monitoring the connections, but sometimes the cloud service provider itself is not a trusted party. In this case we have to protect the data that is on the cloud from the cloud service provider also not only during retrieval but the problem is when we want to answer queries using the uploaded data.

To prevent the above mentioned forms of security challenges, a number of research works have been proposed and various techniques for secure data exchange have been developed. However, most of the existing research works focus on some aspects of the problem, and comprehensive solutions seldom exist. Moreover, a number of these solutions have weaknesses like the efficiency of the algorithm. For instance, the schemes for querying over encrypted data ensure that the cloud service provider is not a threat any more, however, such schemes are not efficient.

To see what scheme suits best as a solution of the problem we need to have some evaluation criteria to compare the available encryption mechanisms. This will also help in figuring out what is needed to be added (if needed) to the best scheme or mechanism we get as a result of the comparison. The following are the evaluation requirements to be considered for the schemes:

1. Data confidentiality: Any unauthorized party (including the cloud server) should not learn any information about the encrypted data files.
2. Fine-grained access control: For users in the same group or different groups, each user can be associated with different access rights which will make the scheme more reliable and efficient as a real life solution.
3. Scalability: The system should have the ability to work efficiently even when the number of authorized user increases.
4. User accountability: When an authorized user becomes dishonest and shares his/her attribute private key with some unauthorized users, he/she should be held accountable.
5. User revocation: If a user quits the system, the user's access rights should be revoked, and the user is denied access to the data. This should be done without effecting other users or needing to change the keys.
6. User rejoin: This refers to the ability of getting the user back into the system after revocation, without effecting other users or keys.
7. Collusion resistant: The system's users should not be able to combine their attributes to decrypt the encrypted data files.
8. Ciphertext size: This refers to the size of the generated file after running the encryption algorithm on the original plaintext data file.

In the following, we provide an overview of some of the most prevalent methods proposed for secure data exchange. These methods are reviewed according to the solution requirements mentioned above.

2. Overview of encryption schemes

In this section we provide a brief overview of some encryption schemes used in cloud data confidentiality management. A number of variants of the following security schemes have been proposed, i.e., most works implement some of these encryption techniques and/or are based on them.

2.1. *AES (Advanced Encryption Standard)*

AES is a Symmetric Key Cryptography algorithm that converts the data files from plaintext format into an incomprehensible format that is called ciphertext which cannot be read by humans to prevent the unauthorized users from gaining access to the data files [1]. In AES, encryption and decryption use the same key which converts e.g. a 128 bit data block to the same size of encrypted content. The key size can be adjusted on per need basis. AES has four main operations/functions: SubByte scrambles each data byte, ShiftRows scrambles the data rows, MixColumns scrambles each data column, and AddRoundKey does the encryption [2].

The approach taken by Prabhakar and Joseph [3] is representative of AES that protects the data for the entire lifecycle from the beginning to the end in the cloud environment. The approach uses AES-256 for encryption along with SSL (Secure Socket Layer) for the sharing and transfer part. The data owner encrypts the data with AES symmetric key encryption to provide data security and then uploads the data to the cloud using SSL. The proposed scheme [3] is claimed to be structured to provide complete security for the data during all stages. It is divided into two phases. Phase one deals with data encryption and secure upload to the cloud. Phase two has to do with the data retrieval which includes authentication process by the data owner and by the cloud service provider and decryption. In the first phase, users register with the data owner so they have the ability to sort and retrieve data files. Authentication process is used here to check for valid users. When the user is authenticated, then that user is provided with a secret key and decryption key. The data owner in this phase also encrypts the data using AES-256 encryption scheme. The encryption is done iteratively and this iteration is based on the encryption key. SSL is then used to protect the data files during the transfer to the cloud. In the second phase, to access the data the user must be authenticated, using a username/password scheme. When authenticated, the user then sends a request to the cloud server. The cloud server verifies the user details and starts the process of data retrieval. When the user receives the data then the data is decrypted in iterations similar to how it was encrypted. The AES encryption scheme protects the data from tampering and for the decryption AES is also used since it is very difficult to guess the key. Since the data remains encrypted and the cloud provider has no knowledge of the key, data confidentiality is guaranteed. For Brute force attacks or exhaustive key searches AES provides ample protection also. However, a major drawback is efficiency and privacy protection.

2.2. *Homomorphic encryption*

In Homomorphic encryption, the plaintext is manipulated with algebraic manipulation like multiplication and addition. These operations behave in a consistent way which means the plaintext is changed according to operations in the ciphertext. This type of encryption is central, and has been in existence since the advent of public key cryptography [4]. There are some known encryption schemes that fall under the homomorphic encryption paradigm such as RSA [5] and Paillier [6]. These schemes are half homomorphic (i.e., support only one operation: either multiplication or addition). Fully homomorphic encryption (FHE) schemes support both operations, i.e. multiplication and addition to the data. In [7] a scheme to perform algebraic query processing over encrypted data is proposed. The proposed scheme is intended to protect the data during all stages of the sharing process based on FHE. It consists of an Evaluate algorithm, KeyGen, Encrypt and decrypt algorithms. Since operators that are normally used in database computation do not work over encrypted data, the paper introduces an extension to the data model. Suppose that a table called R (is in plaintext) with A columns. The scheme represents this as (R', pk) where pk is the encryption public key, and then the table is defined as $R'(A, p)$ with all of A and one more column p which takes the value 0 or 1 to represent the presence of the row in the table. The scheme also provides algorithms for the database operators to work on the encrypted table R'; with bitwise operators, arithmetic and comparison operators, operations on a bit and a word and also implementation for relational algebra. The purpose of these operators is to allow the service providers to perform database operations and get the results while the data is still encrypted. For returning the results to the client, the system cannot simply send the whole result table, as this might be a very large data block. A proposed solution states that the user can specify an estimation of the number of the rows in the output table. The cloud server then sends the output according to the specified number where the sent output has the highest probability of being in the result of the query. This approach can result in a query estimation and not sending the exact query result. The proposed scheme provides an alternative solution by a two-step process for sending the results to the client. In the first step, the service provider computes the sum of the columns of the p value in the result table and this number is sent to the client. The client then decrypts the number (lets call the decrypted number n) then he/she decides on how many rows are needed in the result and asks the service provider for n' rows in the result table. The service provider then sorts the rows of the result on the p while trying to maintain the other order if the query specifies a certain order. The service provider then sends the top n' rows and the client verifies the sum p that should be equal to the sum n. This process provides exact query result to the client. However, a number of issues in the proposed approach are left unanswered, such as practicality of FHE, building indexes, and user authentication.

2.3. *Attribute Based Encryption*

Attribute Based Encryption (ABE) proposed in 2005 contains three main tenants: authority, data owner and data end users (consumers) [8]. Each party has assigned roles in the system. The authority is responsible of generating keys for the data owners and end users to encrypt and decrypt the data files according to certain attributes. The data owner encrypts the data using the keys generated by the authority. The users use their assigned private keys to decrypt the data files in which they are authorized to do so. The attributes of the user trying to decrypt a file are checked and matched with the attributes in the ciphertext. If they do not match, the user is not allowed to decrypt the files even if he/she has the appropriate key.

A secure multi-owner data sharing scheme that supports dynamic groups efficiency where the size and computation overhead of encryption are independent of the number of revoked users is presented in [9]. The focus of the work is to enable sharing data in a multi-owner manner while still preserve data privacy and identity privacy even in untrusted clouds. One basic solution is encrypting the data files and then upload the encrypted data into the cloud. However, this is not entirely practical, due to identity privacy and since any member in the group should be able to manipulate or store the data, and be able to use the services in the cloud. Another reason is that groups are usually dynamic, i.e., users or members in the group change from time to time. These membership changes make it more difficult to secure data sharing. Several approaches or schemes have been proposed for secure data sharing but in these schemes one user stores the encrypted data and distributes the decryption key to the rest of the group members. The problem arises here as the number of data owners and revoked users increase. The technique presented in [9] is able to share and store data files in the cloud, the complexity of encryption is independent of the number of revoked users, user revocation can be done without the need to update the private key for the other users, and a new group member can decrypt stored files directly. Group Signatures are also supported that allow any member of the group to sign messages while keeping the identity secret and it can be revealed by the group manager. Similarly, dynamic broadcast encryption enables the broadcaster to transmit the encrypted data to the users such that only a set of authorized users can decrypt this data. This scheme also allows the group administrator to dynamically add or remove group members while keeping the same old information like the decryption keys and does not need to recompute the keys.

The cloud is operated by cloud service providers (CSP) and provides web services. This entity is not fully trusted by cloud users because usually CSP is not a group member or out of the users' trusted domain. The group manager is in charge of the system and controls system parameters, user registration, user revocation and revealing the identity of data owner. The group manager is fully trusted entity. Group members are the set of the registered users that use the cloud and register their data into the cloud server and share them with the group. To be able to achieve a secure data sharing for the dynamic groups in the cloud, Mona combines the group signature and dynamic broadcast encryption techniques. The group signature enables users to anonymously use the cloud resources and dynamic broadcast encryption allows data owners to share their data in a secure manner. The group manager is responsible for system initialization. To register a user the group manager randomly selects a number and registers the user according to a known equation [9]. For the user revocation the group manager has a public revocation list that is based on which group members can encrypt their data files and ensure the confidentiality against the revoked users.

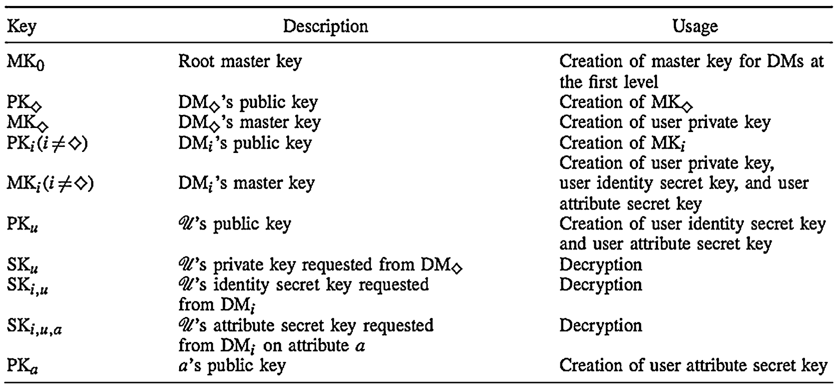
Taeho Jung et al. proposed an encryption scheme based on ABE that provides anonymous privilege control to address the user identity privacy issues [10]. Similarly, a novel patient-centric framework and a suite of mechanisms for data access control in Personal Health Records (PHR) stored in semi-trusted servers is proposed in [11]. The scheme considers a PHR system with multiple PHR owners and users. The owners in this case are the patients and the users are a variety of people like friends or researchers. The scheme supposes the cloud server to be semi-trusted i.e. the server will fetch for secret information in the PHR but follows a general protocol. The problem arises if some users collude with the server to get the information. Another assumption is that each party in the system is preloaded with the public/private key pair. The main idea behind the framework is to divide the system into multiple security domains according to different user's data access requirements. These domains are named public domains and personal domains. The public domains consist of users who have access to the data based on their professional role such as doctors. The personal domains contain users that are associated with the data owner such as family and these gain access rights assigned by the data owner. In both domains, ABE is utilized. The system is noticed to have some limitations for practicality of using ABE and MA-ABE in building PHR systems so a suggestion for future work is to use ABBE (Attribute-Based Broadcast Encryption) scheme to try in resolving the issue. Another thing is the expressibility of the encryptor's access policy is limited by MA-ABE since it only supports conjunctive policies across multiple AAs. The authors believe that there might be a need to use distributed ABE scheme and that might resolve the issue.

A one-to-many (one uploader and several retrievers) encryption system is presented in [12]. The encrypted data file can be decrypted by more than one authorized user or recipient. The ABE-based scheme supports monotonic access formulas that contain AND, OR, or threshold gates. The paper proposes a hierarchical identity based architecture in cloud computing to embody the user hierarchy in the secure cloud storage services sharing. The root private key generator (PKG) delegates the upper level user as the lower level PKG and the use of this is generating the secret keys for all low level users. The secret key transmission is done in a domain for the users to guarantee secure transmission. The sender needs to encrypt the file once only and store just one copy of the ciphertext in a cloud that communicate with no users. This cloud is used by the other users to recover the files using their private keys. The scheme is constructed using bilinear map and is presented using five algorithms. (1) RootSetup: given a large security parameter, the root runs BDH (Bilinear Diffie–Hellman) [13] parameter generator to generate a prime number and 2 groups and then chooses a random number from the first group and chooses 2 cryptography hashes and sets the parameters. (2) DomSetup: is used to generate the secret keys for a certain user by taking the public key as parameter. (3) One2ManyEnc: this is the encryption algorithm that is used to transfer the plaintext into ciphertext such that multiple users can encrypt the data files using their keys. (4) UserDec: this is the decryption algorithm that is used to recover the file by the user given the ciphertext. (5) Recipients-Dec: this is the decryption algorithm which transforms the ciphertext back to plaintext. The thing missing in the scheme is the idea to enable lower-level users to send a short trapdoor to a CSP before retrieving the files. This should prevent information leakage during the transmission process. Yu et al. proposed a scheme based on CP-ABE (Ciphertext-Policy Attribute Based Encryption) to provide the authority with the ability to revoke the attributes of the system users with minimal effort [14]. To resolve the flexibility and scalability issues in ABE schemes, Pandian et al. proposed a scheme based on CP-ABE in 2013 that also provides fine-grained access control for cloud applications [15]. Similarly, Goyal et al. proposed a scheme for secure sharing and storage of cloud data based on KP-ABE (Key Policy ABE) [12].

2.4. *Proxy re-encryption*

Proxy re-encryption was proposed in 1998 by Bluemer, Blaze and Strauss to enable re-encryption of some ciphertext encrypted by one user such that another user will be able to decrypt it [16]. Usually it is used on top of ABE schemes. One of the applications in which proxy re-encryption is useful is when some user wants to forward some encrypted data to another user without the need of key forwarding.

Samanthula et al. proposed a data sharing scheme that is both secure and efficient based on Homomorphic encryption combined with proxy re-encryption [17]. Their main contributions can be summarized as follows. Efficient user revocation: the revocation of the user does not require re-encrypting all the data or new key distribution, Efficient and secure re-join for revoked users: in the case a revoked user wants to rejoin the system either with the same access rights or different access rights, all the data owner needs to do is registering the entry just like registering a new user, preventing collusion between users and CSP: the encrypted data and authorization token list can be outsourced to separate CSPs and hence prevents any collusion between the users and CSP, and Preventing collusion between a revoked user and authorized users: the authorized users can only decrypt the data files in which they are given access rights from the data owner. The proposed framework is a generic scheme where any method in the homomorphic encryption or proxy re-encryption schemes is applicable. The framework consists of five steps as follows: (1) Key Generation and Distribution: here the data owner generates and distributes two types of key pairs based on homomorphic encryption and distributes these keys to the system users. The data owner also generates the proxy re-encryption key for each authorized user. (2) Data Outsourcing: In this stage, the data owner encrypts the data files and generates authorizing tokens for each data file. The next step is uploading the data files to the cloud. (3) Data Access: upon data access request, the CSP checks whether the user is



Key	Description	Usage
MK_0	Root master key	Creation of master key for DMs at the first level
$PK_\diamond$	$DM_\diamond$'s public key	Creation of $MK_\diamond$
$MK_\diamond$	$DM_\diamond$'s master key	Creation of user private key
$PK_i (i \neq \diamond)$	DM_i's public key	Creation of MK_i
$MK_i (i \neq \diamond)$	DM_i's master key	Creation of user private key, user identity secret key, and user attribute secret key
PK_u	$\mathscr{U}$'s public key	Creation of user identity secret key and user attribute secret key
SK_u	$\mathscr{U}$'s private key requested from $DM_\diamond$	Decryption
$SK_{i,u}$	$\mathscr{U}$'s identity secret key requested from DM_i	Decryption
$SK_{i,u,a}$	$\mathscr{U}$'s attribute secret key requested from DM_i on attribute a	Decryption
PK_a	a's public key	Creation of user attribute secret key

Fig. 1 – Keys and their usage [21].

authorized or not and takes the appropriate action accordingly. (4) User Revocation: the data owner performs the actions required for secure revocation of certain user's access rights. (5) User Rejoin: the data owner generates a new authorization token with the desired access rights to the user. The scheme assumes that the user can collude with at most one of the clouds in the system. So for each data record the data owner exports a ciphertext and a list of pairs of proxy re-encryption keys and the corresponding userID for all authorized users to the first cloud denoted as primary cloud and also pairs of the users and ciphertext to the secondary cloud. During the data access, the users send the request to the primary cloud and the cloud does some verification and then sends the request to the secondary cloud that will, in turn, do the homomorphic operations and send the result to the user. So according to the assumption that the user can collude only with one cloud, the collusion will not affect the system and will be useless.

2.5. *Hierarchical Identity Based Encryption*

Hierarchical Identity Based Encryption (HIBE) is an encryption scheme that is used to restrict users who are unauthorized or partially authorized and might share their key with some unauthorized users which will lead to unauthorized data access. Major works in this regard include [18,19], and [20]. HIBE consists of five main steps or operations: Setup generates the public parameters and the master secret, Encrypt takes the plaintext, the public parameter and the set of identities and outputs the ciphertext which is the encrypted content, KeyGen generates the secret key for the provided identity vector w. Decrypt restores the ciphertext into its original plaintext content, and Delegate outputs the secret key for w' which is a concatenation of the identity vector w.

In 2011, Wang, Liu, and Wu aimed to propose a scheme that achieves finegrained access control with a high performance, full key delegation and flexibility. Their proposed approach combines both HIBE (Hierarchical Identity Based Encryption) with CP-ABE (Ciphertext Policy Attribute Based Encryption) [21]. The scenario in which the proposed system is based is a company where the company owner uploads the data on the cloud server and the employees retrieve the data from that server. The scheme has multiple keys and each key has a different usage purpose. These keys and their usage are summarized in the table [21] (see Fig. 1).

The proposed scheme consists of seven algorithms with polynomial run-time. Setup: this algorithm takes a large number of security parameters (k) as input and then outputs system parameters and the root master key MK0. CreateDM: this algorithm is used to generate the Master keys for the DMs by using params and the master key. CreateSK: this algorithm is used to generate the private key for the user using param and the master key if and only if the user is eligible. If not, then it will output "NULL". CreateUser: this algorithm generates the secret key of the userID as well as the user attribute secret key. Encrypt: this is the encryption algorithm to transfer the message from plaintext to ciphertext. RDecrypt: this is the decryption algorithm that decrypts the ciphertext to original plaintext message using the user's private key if the userID belongs to the recipients set. ADecrypt: this is another decryption algorithm that depends on the users attributes (whether they satisfy the jth conjunctive clause in the attribute based access control policy). The scheme is proved to be secure and at the same time collusion resistant. However, an open issue in the scheme or something for future scheme enhancement is implementing more expressive encryption scheme so we can have full security under standard model but enhance the performance of the scheme.

In 2013, Dong et al. proposed "SECO" [22] which is a secure and efficient collaboration scheme based on HIBE to ensure the data confidentiality on the untrusted clouds. The scheme employs a two-level HIBE to ensure the confidentiality of the data files in untrusted clouds. The proposed scheme is considered the first attempt to explore the secure data collaboration service which prevents leakage and enables one-to-many encryption. It also enables data writing operation and fine-grained access control. SECO realizes one-to-many encryption paradigm such that the encrypted domain data can have many authorized users that are able to decrypt the data files. The private key generator (PKG) manages a number of D-PKGs (domain private key generator) while D-PKG manages a number of domain users. The data owner encrypts the data with multiple users in the domain using the public key and stores the data on the cloud server. Users outside the recipients list will not be able to decrypt the data or even learn any information from the data files. The following is the set of algorithms or steps that are the components of the scheme. Root Setup: The R-PKG (root private key generator) takes a security parameter K as input and outputs the system parameters a root master key. Domain Setup: Each D-PKG obtains the system parameters from the R-PKG and randomly picks a master key that will be used to generate the private keys for the domain users. Key Generator: R-PKG uses its master key to generate private keys for D-PKGs while D-PKGs use the system parameters and their secret keys to generate the private keys for all domain users. Encryption: The data owner inputs the system parameters, plaintext message and ID-tuples of the intended data authorized users to generate the ciphertext. Decryption: The user or D-PKG inputs the system parameters, ciphertext and the private key to recover the original plaintext data from the encrypted data file.

2.6. *Identity Based Broadcast Encryption (IBBE)*

One of the earliest works on Broadcast Encryption was presented by Naor and Fiat in 1994 [23]. In Broadcast Encryption, the data broadcaster encrypts the data file and sends it to a group of users. These users then use their private keys to decrypt the message. The broadcaster selects a set of identities at the encryption step, so that only the intended users are able to decrypt the transmitted file(s). This encryption scheme is collusion resistant [24]. An example of an efficient and secure data sharing technique based on Identity Based Broadcast Encryption (IBBE) and Public key Encryption with Keyword Search (PKES) is presented in [25]. The construction has a two layered access control on shared encrypted data. In the first layer, there are two mechanisms provided which are defined for authorization and revocation, so each user will be able to get a private key to perform search and decryption. Moreover, if the user's key is revoked, it will not be able to read and search the data. The second layer provides an identity based access control mechanism, so that users will be able to upload encrypted data and submit a set of identities along with it such that the users in the set are the only users who have access to the data. The construction also provides a keyword based search so that authorized users can use their private keys to generate queries. The scheme consists of seven algorithms. A brief overview follows. Let U be the user Identity universe and M be the message, then we have the following algorithms:

- Setup(λ, n): executed by the authority. Input is λ which is a security parameter and n is number of system users. Outputs the master key MK and the public key PK.$\mathrm{SK}_{\mathrm{SIG}}$
- Extract(MK, ID_i): executed by the authority. Input is MK and ID_i which represents the identity that is when receiving private key request on it, the algorithm will output the user's private key SK_i.
- Encrypt(S, M, W): executed by the data owner. Input is S which is the set of authorized users, M is the message and W is the set of keywords. Output is the encrypted content CT (ciphertext).
- Query(SK_i, ID_i, W): executed by the user. Input is the user's private key SK_i, user's identity ID_i and the keyword W. Output is the user's query Q_i, w.
- Trapdoor(MK, Q_i, w): executed by the trusted private cloud. Input is master key MK, user's query Q_i, w. Output is the trapdoor $\mathrm{TP}_\mathrm{i}, w$.
- Test(CT, $\mathrm{TP}_\mathrm{i}, w$): executed by the public cloud server to search the matched ciphertext. Input is the encrypted content CT and the trapdoor. Output is yes or no where yes means that the trapdoor matches the ciphertext.
- Decrypt (PK, $\mathrm{SK}_\mathrm{i}, S, hdr$, CM): executed by the user. Input is the public key PK, the user's private key SK_i and the returned ciphertext (S, hdr, CM). Output is the original message M in plaintext.

In the initialization stage, the authorized party runs the Setup algorithm to get the public key and the master key. Then in the data upload stage, any authorized user will be able to submit the set S along with the encrypted data. In this stage, the user runs the Encrypt algorithm to encrypt the data prior to uploading. In the new user grant stage, a new user needs to apply to get the private key and uses the Extract algorithm. During the Data Access stage, the user runs the Query algorithm to generate the query. Upon receiving the query, the trusted private cloud runs the Trapdoor algorithm to generate the trapdoor TP which is then used for the Test algorithm that tests the ciphertext CT and returns the matched ones to the user. Finally, the Decrypt algorithm is used to convert the ciphertext back to the original plaintext message. As for the performance of the method, some stages take constant time regardless of the number of the system users like the initialization stage, but there are some stages that take more time as the number of the users increase. However, in the existing analyses of the scheme, both the verification and decryption are not considered.

3. Other solutions

Some other techniques to address the security issues in cloud data sharing have also been proposed. Some of these provide their own encryption mechanisms like the solution proposed in [26] which enables authorized users to perform keyword-based search directly without sharing the secret key. The proposed solution provides two layered access control to limit unauthorized access to the shared data. It assumes a trusted private cloud and public cloud in the system. Moreover, the system construction does not rely on shared keys. Major features are summarized below:

- Two-Layered Access Control: is performed on the shared encrypted data. The first layer takes care of the authorization and revocation mechanisms, where each user obtains an individual private key for searching and decryption. The second layer provides an identity based access control mechanism where the users are allowed to submit identities along with uploading the encrypted data.
- Keyword-Based Search: is performed only by authorized users that are able to use their private keys to generate the queries.

The scheme may seem similar to other prevalent schemes algorithmically. However, some differences at various stages can be noted. An overview of the specific steps or phases of operations performed by the scheme, and associated algorithms follow.

- Initialization: On system initialization, the authority will run the setup algorithm to obtain the keys. The difference here between this scheme and other schemes is that the setup algorithm used to initialize the system generates the PK and MK like the previous ones but MK here consists of two parts MK_{IBBE} and MK_{PEKS}. PK will be published where the two parts of MK will not.
- Data Upload: This step is the data upload phase where the data owner will run encrypt algorithm to encrypt the data prior to uploading the files to the cloud.
- New user Grant: The new user will apply to the authority for the private key and the authority will run Extract algorithm using MK and the user ID. Again this scheme will divide the key SK into two parts SK_{IBBE} and SK_{SIG}.
- Data Access: In the current approach, this step is a bit more complex than the previous ones. The user will run the Query algorithm to generate the query and send it to the cloud. When the cloud gets the query, the server will run the Trapdoor algorithm using SK_{PEKS} part of the key to generate a trapdoor that will then be used in the Test algorithm that performs tests on the ciphertexts and returns the matched ones to the user. The user then needs to decrypt the ciphertext to get the original message.
- User Revocation: For user revocation, just the corresponding registration information is deleted.

Another class of solutions provide a software on the client's side that handles transferring the data securely. A prime example is that proposed in [27] which is a new architecture for secure applications in which the only software running on the end host is a light-weight secure thin terminal, and most application logic resides at a remote cloud rendering engine. The scheme uses the enduser's workstation as a secure I/O path to access application logic in the cloud. This allows for a radically client-side platform so the client-side STT (Secure Thin Terminal) needs only to render graphical data from a remote application and forward input events to it. The STT is isolated from the user's untrusted operating system through a small layer that is called microvisor. A cloud rendering engine CRE executes an application that produces a bitmap image to appear on the user's screen and then sent to the STT over an encrypted protocol.

The approach centers on two abstractions: the STT on the client side and the CRE on the cloud server. The STT is a software that runs on the user's workstation to provide secure access to a remote application, without the need of trust in any other software on the workstation. The CRE is the server side counterpart of the STT and it contains almost all the functionality of the application and runs an isolated instance of the application for each STT on a separate virtual machine. These virtual machines run a minimal software stack needed to render the remote application rather than allowing the user to install his/her own software.

The main problem with such solutions is the need of installing the software on the client's workstation and trying to find an alternative solution that is cloud-based. This is extra work for the clients who want to use the solution and at the same time it limits the range of the supported devices. Another proposed solution uses PaaS (Privacy as a service) to solve the problem of processing sensitive data in cloud infrastructures securely [28]. The main goal of the design of PasS is to maximize the user control in managing the various aspects related to the privacy of the sensitive data files. This goal is achieved by implementing a user-configurable software and data privacy categorization mechanism. The scheme also provides the users with feedbacks on the operations performed on their data files.

The proposed scheme relies on cryptographic co-processors to provide isolated secure processing containers. The co-processor is a piece of hardware card that interfaces with the server through PCI-based interface. The crypto co-processor resists physical attacks because of the tamper-proof casing that encloses it. The organizational unit is responsible of crypto co-processor configuration and distribution to the cloud service providers. It needs to be installed on every server running a virtual machine for the users that are registered in the privacy service. The system user is responsible of configuring his/her software applications to support security mechanisms enforced by the service. It is possible to run the application in a secure co-processor without applying software divisions, but doing so will affect the performance and efficiency of the application. Prior to uploading the data to the cloud server to be stored and processed, the user needs to classify the data based on sensitivity and significance into the following categories:

1. NP (No Privacy): the data files that are not sensitive, i.e. in which the cloud service provider is fully trusted to store the data with no encryption.
2. PTP (Privacy with Trusted Provider): the service provider is trusted to encrypt the data files.
3. PNTP (Privacy with NON-Trusted Provider): the user encrypts the data files prior to uploading them to the cloud server.

The scheme also has three privacy protocols that are:

- Data and Software Transfer Protocol.
- Software Execution and Data Processing Protocol.
- Privacy Feedback Protocol.

A major drawback of the proposed scheme is its reliance on a hardware piece that needs to be installed on the server for the scheme to function, thereby minimizing the practicality of the solution.

4. XACML for cloud data security

XACML is a markup language that is based on XML and used to enforce access control. It is an OASIS framework that specifies and enforces the rules for the access control. There are some extended versions of XACML that provide easy to use data sharing that is secure, flexible and scalable [29]. XACML became a standard that is used for enforcing the access control and the extended versions are providing fine grained access control [30]. Since it is based on XML, it is easy for humans to read and understand it, and it follows the well-formed XML rules. Khadilkar et al. addressed the data sharing issue in their paper [31] by combining cloud computing technologies with XACML policy based security mechanisms to provide fine-grained access control to cloud resources. The following is a description of their system layers and components:

- Web application layer: this is the interface of the system in which the cloud users access the infrastructure. This application provides an authentication mechanism as a login page that uses Java Simplified Encryption to store the usernames and passwords in a file that is not accessible to any user.
- The ZQL parser layer: this layer takes any query submitted by the user and if no error is returned, it proceeds to the XACML policy evaluator. This parser is an SQL parser that is written in JAVA.
- XACML policy layer: this layer is divided into subsections that are defined for the framework. XACML Policy Builder defines role based access control policies on the resources by defining a mapping in the query between users and the query type. Moreover, the policy evaluator is used during query execution and also during view creation.
- Basic query rewriting layer: this layer enables adding another abstraction layer between the user and HiveQL by allowing the user to input SQL queries that are rewritten in HiveQL syntax.
- Hive layer: this is the data warehouse that is built on top of Hadoop and provides the ability to structure the data in HDFS as well as the query.
- Hadoop Distributed File System (HDFS) Layer: this is a distribute file system designed to run on basic hardware and stores data files according to tables created in Hive.

The system was implemented on a 19 node cluster with a mix of two different configurations for nodes. The experiments used two data sets to test the systems performance versus Hive.

The proposed method suffered from the lack of some keyword based groups such as UPDATE and DELETE. Another issue with it is that it needed to be extended to public clouds. Anh et al. also proposed an XACML based scheme to support secure and flexible data sharing framework for cloud that extends XACML model by integrating flexible access control decisions in [30]. They demonstrated the need of a secure flexible data sharing scheme, extending the XACML framework to support fine-grained policies and implemented a prototype of the framework providing a user friendly user interface and evaluating the prototype's performance. Their proposed prototype needed some improvements such as minimizing the number of data servers, support stream databases and continuous queries and most importantly providing more fine-grained access control by relaxing the trust assumptions.

5. Logging and auditing

Encryption schemes (as the ones mentioned above) usually do not offer user accountability [32]. In these cases, some form of logging or auditing mechanism is needed. Auditing schemes usually consist of two main components: the logger and the log harmonizer. The logger records the system log in a file (e.g. JAR) and the log harmonizer is responsible of providing the log files to the system manager/data owner, to take further actions if needed [33]. The following is an example of a scheme that clarifies the auditing and logging mechanism.

The Cloud Information Accountability (CIA) system [34] performs automated logging and distributed auditing for relevant access that are performed by entities in the cloud system. This is supposed to add more security to the cloud infrastructure. The CIA consists of two major components, these are the logger an log harmonizer. A JAR file contains a set of rules for access control to specify the authority of each party in the data access. The integrity of JRE on the system is also going to be checked on which the logger is initiated via oblivious hashing. The scheme converts the JAR file into obfuscated code for more infrastructure security. The main problem here is when the cloud asks for user's personal and confidential data that are sometimes necessary to perform a certain task. This data need to be secured from sharing with a third unauthorized party. The issue here is not sharing the data only but also at certain points both the data owner and the cloud service provider might loose control over the data while the data is processed in a chain of interaction between parties. The system design is as follows. JAR Generation: This JAR file contains the set of access roles. The JAR file provides usage control to perform logging based on the configuration settings that were defined at the creation time. Obfuscation: Obfuscation in software development refers to creating obfuscated code deliberately that is the machine language that cannot be read by humans. This is used to avoid tampering and reverse engineering to get the source code. The code is generated using obfuscator that are programs to convert the code into obfuscated code. Logging Mechanism: The JAR file is responsible of handling authentication of entities which want to access the data. Log Record Generation: Log records are generated using the logger component. Logs are generated whenever some party tries to access the data and these logs are appended to the JAR file. Provable Data Possession (PDP): The PDP system can be generated in two phases, setup and challenge. This provides security and catches different kinds of attacks. Auditing Mechanism: Data owners will get a frequent update of the access record to their data. Accountability Mechanism: After the log file is sent to the data owner by the log harmonizer, the data owner can then check the log file and take appropriate actions. Open Issues include adding the JRE verification mechanism to the scheme and enhancing the PDP architecture at the user-end to allow an efficient usage for the PDP.

Another scheme that conducts automated logging and distributed auditing of relevant access performed by entities in the system was proposed in [35]. The enhancement in this scheme compared to previous versions is taking concern of the JAR file by converting it into obfuscated code to add more security to the infrastructure. The scheme's idea is similar to previous ones in the way it functions with just a little modification done to it. The main components of the scheme are also the same, and these are the logger and the log harmonizer which perform the same operations performed by previous schemes. The main difference or modification that was done is converting the JAR file to obfuscated code to add more security to the framework infrastructure. The scheme was implemented by setting up a small cloud and the test environment consisted of several Open SSL-enabled servers.

Although that the proposed scheme improved the security of existing schemes that perform automated logging, the

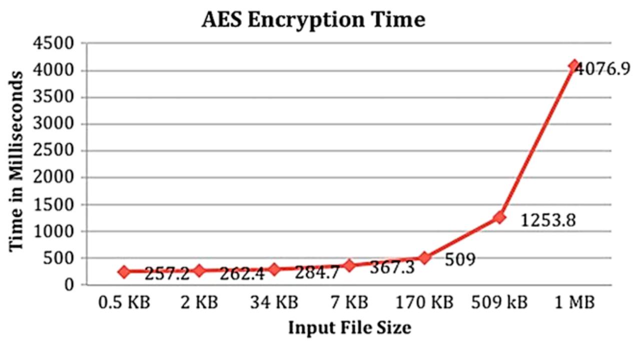


Fig. 2 – Performance of AES encryption.

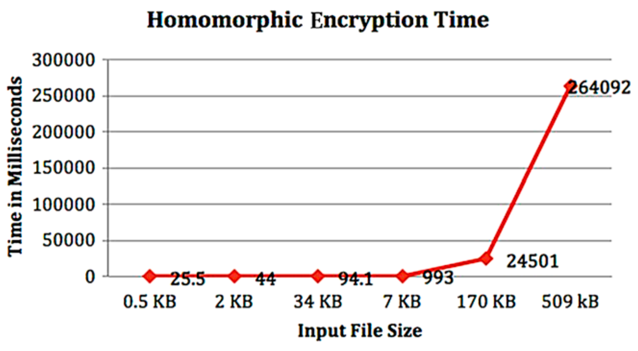


Fig. 3 – Performance of homomorphic encryption.

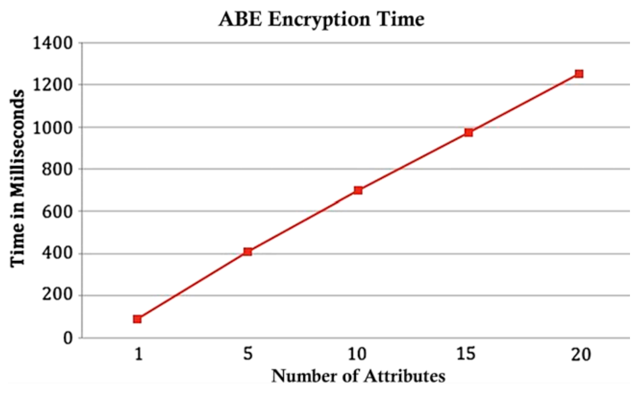


Fig. 4 – Performance of attribute based encryption.

scheme still has the same problem (like the existing ones) of having a mechanism to verify the integrity of the JRE and the suggested solution up to the time this paper was written is to leverage the advantage of secure JVM that is developed by IBM. Another is enhancing the PDP architecture from the user side to allow efficient system usage.

6. Experiments

A number of schemes have been proposed for secure cloud data sharing. In this section we present a brief comparison of some of the encryption schemes discussed earlier. We ran the proposed schemes on a small application (JAVA-based) for comparison purposes. The application takes plaintext files as input and provides encrypted files along with the encryption time as output. The computer used on the execution was a mac running OS X on 2.5 GHz Intel core i5 processor and 8 GB 1600 MHz DDR3 Memory. In these experiments we provided some files with different sizes to be encrypted using different encryption schemes. The code has a timer that gives the exact time from right after reading the file until right before starting the writing (this period is where the encryption process happens). We performed repeated tests to get averages and to eliminate errors. One of the main things we got out from the experiments other than the encryption performance is the size of the encrypted file after getting the ciphertext from the plaintext file. This is an important parameter since the larger the file, the more the overhead.

The comparison between some of the major available encryption schemes according to the evaluation criteria discussed earlier is shown in Fig. 6. The table compares the schemes based on the security requirements mentioned above as well as the encryption time complexity, which is the time, needed to encrypt the file where X indicates the existence of the criteria and x indicates the absence of it. ACT is the attributes associated with the data, G1 and G2 are two bilinear groups of prime order p. In FHE, t is the security parameter. In HIBE l is the levels of the Hierarchy and T is Time Slices. Ciphertext size in the table refers to the size of the file that is generated as a result of running the encryption process on the plaintext (input file) (see Fig. 2).

Figs. 3 through 6 show the performance of encrypting the files using the mentioned encryption schemes:

We can clearly see from the figures that AES starts with a very short encryption time but as the file size grows, the time increases rapidly. We can thus concluded and that AES works best with small input files. For Homomorphic Encryption, it starts with a higher timing than AES for small files, but then at larger files it gives the results faster than AES which makes it better for large files in terms of performance. Attribute Based Encryption, as can be seen from the figure is affected by the number of attributes associated with the file to be encrypted. As the number of attributes grows, the time needed for encryption also increases. Similarly, for Hierarchical Based Encryption the file size gives approximately a linear increase in the encryption time, but it is mostly affected by the number of the levels in the hierarchy.

7. Conclusion

Cloud computing is an emerging paradigm, and security is the most important factor in cloud data sharing since in many cases the data being shared is sensitive and unauthorized access might be harmful to the data owner. This requires finding a secure data sharing scheme to deploy in the environment where the data is being shared. Many schemes have been proposed in the literature depending on encryption mechanisms to provide data security. In this paper, we have presented a survey of major secure data sharing schemes for the cloud environment. The focus of the survey is to show

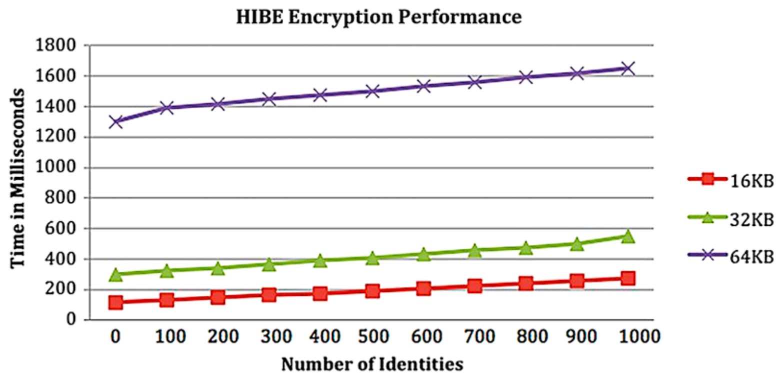
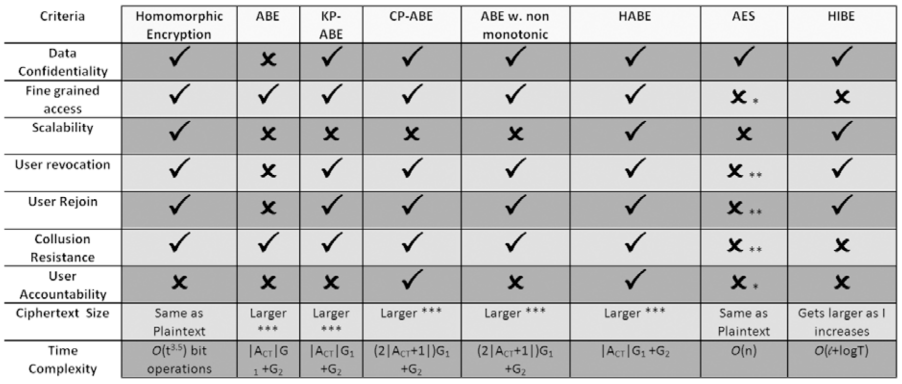


Fig. 5 – Performance of HIBE.



Criteria	Homomorphic Encryption	ABE	KP-ABE	CP-ABE	ABE w. non monotonic	HABE	AES	HIBE
Data Confidentiality	✓	✗	✓	✓	✓	✓	✓	✓
Fine grained access	✓	✓	✓	✓	✓	✓	✗ *	✗
Scalability	✓	✗	✗	✗	✗	✓	✗	✓
User revocation	✓	✗	✓	✓	✓	✓	✗ **	✓
User Rejoin	✓	✗	✓	✓	✓	✓	✗ **	✓
Collusion Resistance	✓	✓	✓	✓	✓	✓	✗ **	✗
User Accountability	✗	✗	✗	✓	✗	✓	✗ *	✗
Ciphertext Size	Same as Plaintext	Larger ***	Larger ***	Larger ***	Larger ***	Larger ***	Same as Plaintext	Gets larger as l increases
Time Complexity	$O(t^{3.5})$ bit operations	$\|A_{CT}\|G_1+G_2$	$\|A_{CT}\|G_1+G_2$	$(2\|A_{CT}+1\|)G_1+G_2$	$(2\|A_{CT}+1\|)G_1+G_2$	$\|A_{CT}\|G_1+G_2$	$O(n)$	$O(\ell+\log T)$

Fig. 6 – Comparison table according to the evaluation criteria. Symbols in the table : * Usually combined with other tech. to provide it. ** Since same key is used for Encryption/Decryption. * Since attributes are added to the encrypted content then the size will grow depending on the number of attributes. + Need to Encrypt again removing the user ID. So has encryption overhead.**

how encryption is used in each of the covered technique, and to discuss the corresponding open issues (if any). The brief survey also provides a comparison to make discussion clear. From the comparison table we could infer that the best schemes (schemes that cover the most security requirements) are Fully Homomorphic Encryption (FHE) and Hierarchical Attribute Based Encryption (HABE). In our future work, we aim to propose a scheme that will contain the security features in these while overcoming any deficiencies/open issues in them.

REFERENCES

[1] A. Tripathi, M.S. Jalil, Data access and integrity with authentication in hybrid cloud, Orient. Internat. J. Innovative Engrg. Res. 1 (1) (2013) 030.

[2] N. Kangude, P. Wani, S. Raut, Advanced encryption standard.

[3] D.M. Prabhakar, K.S. Joseph, A new approach for providing data security and secure data transfer in cloud computing.

[4] M.G. Kaosar, R. Paulet, X. Yi, Fully homomorphic encryption based two-party association rule mining, Data Knowl. Eng. 76 (2012) 1–15.

[5] R.L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Commun. ACM 21 (2) (1978) 120–126.

[6] P. Paillier, Public-key cryptosystems based on composite degree residu- osity classes, in: In Advances in cryptologyEUROCRYPT99, Springer, 1999, pp. 223–238.

[7] M. Mani, K. Shah, M. Gunda, Enabling secure database as a service using fully homomorphic encryption: Challenges and opportunities. arXiv preprint, 2013. arXiv:1302.2654.

[8] P.-S. Chu, C.-C. Lee, P.-S. Chu, P.-S. Chung, M.-S. Hwang, et al., A survey on attribute-based encryption schemes of access control in cloud environments, 2013.

[9] X. Liu, Y. Zhang, B. Wang, J. Yan, Mona: secure multi-owner data sharing for dynamic groups in the cloud, IEEE Trans. Parallel Distrib. Syst. 24 (6) (2013) 1182–1191.

[10] T. Jung, X.-Y. Li, Z. Wan, M. Wan, Privacy preserving cloud data access with multi-authorities, in: In INFOCOM, 2013 Proceedings IEEE, IEEE, 2013, pp. 2625–2633.

[11] M. Li, S. Yu, Y. Zheng, K. Ren, W. Lou, Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption, IEEE Trans. Parallel Distrib. Syst. 24 (1) (2013) 131–143.

[12] Q. Liu, G. Wang, J. Wu, Efficient sharing of secure cloud storage services, in: Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on, IEEE, 2010, pp. 922–929.

[13] D. Boneh, M. Franklin, Identity-based encryption from the weil pairing, in: Advances in CryptologyCRYPTO 2001, Springer, 2001, pp. 213–229.

[14] S. Yu, C. Wang, K. Ren, W. Lou, Attribute based data sharing with attribute revocation, in: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ACM, 2010, pp. 261–270.
[15] C.M. Pandian, K. Seenivasan, Flexible and fine-grained access control in cloud computing using hierarchical based encryption scheme, Internat. J. Engrg. 2 (4) (2013).
[16] K. Fu, S. Hohenberger, proxy re-encryption, 2005.
[17] B.K. Samanthula, G. Howser, Y. Elmehdwi, S. Madria, An efficient and secure data sharing framework using homomorphic encryption in the cloud, in: In Proceedings of the 1st International Workshop on Cloud Intelligence, ACM, 2012, p. 8.
[18] J.H. Seo, J.H. Cheon, Fully secure anonymous hierarchical identitybased encryption with constant size ciphertexts. IACR Cryptology ePrint Archive, 2011:21, 2011.
[19] J. Baek, J. Newmarch, R. Safavi-Naini, W. Susilo, A survey of identitybased encryption, 2005.
[20] S.K. Parsha, Mohd. Khaja Pasha, Enhancing data access security in cloud computing using hierarchical identity based encryption (hibe), Internat. J. Sci. Engrg. Res. 3 (5) (2012).
[21] G. Wang, Q. Liu, J. Wu, Achieving fine-grained access control for secure data sharing on cloud servers, Concurrency Comput. Pract. Exp. 23 (12) (2011) 1443–1464.
[22] X. Dong, J. Yu, Y. Luo, Y. Chen, G. Xue, M. Li, Achieving secure and efficient data collaboration in cloud computing. In Quality of Service (IWQoS), in: 2013 IEEE/ACM 21st International Symposium on, IEEE, 2013, pp. 1–6.
[23] A. Fiat, M. Naor, Broadcast encryption, in: Advances in CryptologyCRYPTO93, Springer, 1994, pp. 480–491.
[24] C. Delerablée, Identity-based broadcast encryption with constant size ci-phertexts and private keys, in: Advances in Cryptology–ASIACRYPT 2007, Springer, 2007, pp. 200–215.
[25] J. Li, J. Li, Z. Liu, C. Jia, Enabling efficient and secure data sharing in cloud computing, Concurrency Comput. Pract. Exp. 26 (5) (2013) 1052–1066.
[26] J. Li, J. Li, Z. Liu, C. Jia, Enabling efficient and secure data sharing in cloud computing, Concurrency Comput. Pract. Exp. 26 (5) (2014) 1052–1066.
[27] L. Martignoni, P. Poosankam, M. Zaharia, J. Han, S. McCamant, D. Song, V. Paxson, A. Perrig, S. Shenker, I. Stoica, Cloud terminal: Secure access to sensitive applications from untrusted systems, in: USENIX Annual Technical Conference, 2012, pp. 165–182.
[28] W. Itani, A. Kayssi, A. Chehab, Privacy as a service: Privacy-aware data storage and processing in cloud computing architectures, in: In Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference on, IEEE, 2009, pp. 711–716.
[29] B. Parducci, H. Lockhart, E. Rissanen, Extensible access control markup language (xacml) version 3.0. OASIS Std., August, 2011.
[30] D.T.T. Anh, W. Wenqiang, A. Datta, City on the sky: Flexible, secure data sharing on the cloud, 2011.
arXiv preprint arXiv:1108.3915.
[31] V. Khadilkar, A. Gupta, M. Kantarcioglu, L. Khan, B. Thuraisingham, Secure data storage and retrieval in the cloud, University of Texas, 2011.
[32] S. Naqvi, S. Naqvi, S. Khan, S. Malik, Application specific scalable architectures for advanced encryption standard (aes) algorithm, WSEAS Trans. Electron. 5 (10) (2008) 427–436.
[33] M. Epuru Madhavarao, C. JayaRaju, Data sharing in the cloud using distributed accountability.
[34] T. Praveenkumar, K. Narsimhulu, Secure and accountable data sharing in the cloud.
[35] S. Sundareswaran, A.C. Squicciarini, D. Lin, Ensuring distributed accountability for data sharing in the cloud, IEEE Trans. Dependable Secure Comput. 9 (4) (2012) 556–568.