Comparative Study of Privacy Preservation and Access Control of Cloud Data

Disha M. Wagle
Department of Computer Engineering
MCT's Rajiv Gandhi Institute of Technology
Mumbai, Maharashtra

Abstract— **Security of data involved during sharing of data in a cloud computing environment is one of the biggest concerns in a cloud platform. Nobody should be trusted with important data, not even the Cloud Service Provider. Even though the Cloud Service Providers enforce the access control policies and follow protocols, they are 'curious'. They want to find out as much information about the user's data as possible. There have been several approaches which have been adopted in the past, to provide protection to the data but most of these techniques have certain disadvantages. Hence, a useful approach has been devised. This approach makes use of multiple layers of commutative encryption to protect data against Cloud Service Providers while an authorization mechanism enforced by the Cloud Service Provider is responsible for data protection against unauthorized users.**

Keywords—Cloud, privacy preservation, access control, encryption, Cloud Service Providers

I. INTRODUCTION

Cloud computing is one of the most sort after technologies in today's time. There are several concepts which have contributed to the facilities it provides and the major cause of its success. Some of them are virtualization, utility computing, elastic computing and network centric content [1]. The cloud makes various provisions to its users like resource pooling, pay as you go, on-demand self service and elasticity. Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), also known as the SPI framework is the service provided by cloud [2]. But of late, Data as a Service has been increasingly growing mostly due to the large data storage and retrieval requirements of users. The data is stored in the cloud as well as processed within the cloud and has the ability to be shared with multiple users at the same time. The cloud provides a virtual environment which makes its memory scalable. The data needs of a big, medium or small scale enterprise may be satisfied by the cloud. But along with every advantage a compromise has to be made. When it comes to cloud, security is a major concern.

Security can be viewed with respect to several aspects. According to [3], security risks could be traditional security risks, risks due to system availability and risks due to third-party data control. In this paper, we have concentrated on one of the most important concerns which arise due to third-party control of data and cloud service providers (CSP). Privacy preservation of data and access control to the cloud has been a growing concern. Before a user starts using the cloud services, he/she has to provide certain information about himself/herself. The user also agrees to a contract with the CSP which, usually, makes an explicit statement that the cloud providers are not responsible for any data loss or leakage and that the user should store the data at his/her own risk. This further increases the concerns. But most of the cloud providers store data on their servers in order to recover from an accidental data loss. This could lead to leakage of data from the servers. These methods are adopted by the CSPs without the knowledge of the user hence if a user deletes some data from the cloud, the data might still exist on the server, probably upon replication failure. A large-scale enterprise having its private cloud can probably store all its confidential information like its financials onto the cloud. A rogue employee could mess with these details since the CSP has the ability to access the user data if security is not appropriately implemented [1].

Cloud Service Providers are curious about the data stored by the users. The CSP could provide customer information to a third party like advertising company to target individuals for advertisements. Hence, it is extremely important that the the privacy of the users be maintained and the data stored is provided the right kind of access control.

The paper focuses on the various systems proposed in order to maintain privacy and how different kinds of access control policies are devised to control authorization to individual resources provided by cloud service. A comparative study is provided for the analysis of the systems. This shows how the systems have gotten better over a period of time taking into consideration several new parameters every time.

II. PRIVACY AND ACCESS CONTROL MODELS

There have been several systems which have been proposed, specifically, to address the needs of privacy preservation of user data and control the access multiple users might have to the cloud data.

A. RBTBAC MODEL

Model: According to [4], Role Based Time Bound Access Control Model considers Electronic Health Record datasets. This system introduces the role based access. Every user has some role, which could be that of a doctor, patient or staff. Each and every role has some privilege depending on the personal details of the user. This role based model is constructed using a hierarchical approach. Also, the access control policies which are devised for every user involve the time parameter. A user with a certain role, along with selective privileges, also has a time constraint which is the authorized time interval within which the user can access the records.

The encryptor collects health record information from various organizations and institutions. The data is encrypted and stored. When a new user wishes to gain access to the data, the Trusted Authority has to be contacted. In order to gain access to the records a new user should get hold of the system identity credential and access credential. The system identity credential includes an ID, role of user, user's master key and some system parameters. Once this credential is acquired, the user can log in to the system but not access the records. Then an access request is made which includes the user's role, patient's ID, access time period and significant words pertaining to data needed. The TA examines the request and then gives access to the user. During the specific time period, if user needs access to some other data then new system identity credentials are not required.

The patient's privacy is maintained too, as the patient is involved in deciding the access control policies as to who can access what data related to his information. The policy for a patient can be expressed as follows: policy=(r, < P_j, {o} >,< tb, te, ti >, mo, op). Here, r is the user's role, P_j is the patient whose data is needed along with the target data to be accessed, tb, te and ti is the start time, end time and the maximum time data can be accessed, respectively, mo is the motive of accessing the data and op is the output of access request, whether or not access is given.

For the EHR, a tree is constructed wherein each patient is the root. The tree consists of index nodes, i.e. nodes for a doctor and nodes according to the doctor's specialization and hospital. Data nodes are also present in the tree, these are the leaf nodes which are which contain diagnosis information for the patient.

Implementation: The RBTBAC is implemented in several steps. These steps are:-

1. Initialization: Here, the encryptor chooses a random value, R, to calculate the root node for the hierarchical tree structure. It calculates the master key which is the root node, $K_{root} = g^{H'(r)}$. Similarly, it calculates keys for each class or particular set of data. Here, g is the generator of a collision-resistant hash function. Also, the encryptor encrypts the index nodes with the class key. Hence, the system parameters are initialized.

2. Encryption: The RBTBAC model puts a time constraint for access to EHR. The time granules are represented in the form of a complete binary tree. Each leaf node of the tree denotes the value of the smallest time granule while the root node denotes the entire timeline. In order to encrypt a class of data for its access only at a particular time, access key for each class is calculated which includes the class key and the value of time interval. $K_{k,t} = H(K_k \| V_{B(t)})$, where K_k is the key to class Ck and $V_{B(t)}$ is the value of time interval t.

3. User Registration: When a new user (Doctor D) is wants to access the EHR, then the system identity credential given by TA contains the master key, KD. The credential includes sc{$enc_{pkd}(K_D,H(.))$} which means the cipher text is encrypted using the private key algorithm with D's public key and this encrypted form is submitted to the user in the form of a digital signature with the TA's public key.

4. User Request: After the doctor gains the system identity credential, the TA verifies the request, searches for the access control policies and issues an access credential to the doctor which includes the encrypted forms, using the data's private key, of the set: begin time, end time, root nodes of the full binary trees and relationship values for the concerned class. The root nodes of FBSs is calculated using the following algorithm:

```
procedure FindRootOfFBS
level = 0, head = tb, end = te;
while head < end do
    if (head mod 2 == 1) then
        RootNodes ! elem:val = head;
        RootNodes ! elem:pos = level;
        RootNodes = RootNodes ! next;
        head=head + 1;
    end
    if (end mod 2 == 0) then
        RootNodes ! elem:val = end;
        RootNodes ! elem:pos = level + 1;
        RootNodes = RootNodes ! next;
        end = end □ 1;
    end
    level + +;
    head = head >> 1;
    end = end >> 1;
end
Print(RootNodes);
```

5. Access and Decryption: The user, thereafter sends a request of access to data. The user's credentials are verified and the required data is retrieved. After this, the user D, computes decryption key $K_{k,t}$ which decrypts the data.

Advantages: The various plus points for this system are-

- The master key for every user is never known to anybody except the user itself. Hence security is maintained.
- Attack by an unauthorized user, attack on an unauthorized class, attack in an unauthorized time interval or a collusion attack is not possible due to the lack of the digital signature of the Trusted Authority which is crucial for authentication during data retrieval.
- The privacy of patients is maintained too since the patients are involved in construction of the access control policies.
- Also, a credential revocation list is used which store the serial numbers of all the expired credentials. This list is

checked while deciding whether access is to be given or not. Hence, credentials issued expire after a certain time and need to be revocated. These expired ones cannot be reused.

- The hierarchical tree key management scheme saves space as it has a space complexity of $O(N_{leaf})$ as compared to traditional techniques which have a space complexity of $O(N^2-N)$, where N is the total number of nodes and N_{leaf} is the number of leaf nodes.
- During encryption key generation for a specific time granule, the time granule has to be calculated which is done using one-way hash evaluation. Hence the time complexity of the number of hash evaluations is less than $\log_2(z+1)$ times less using the binary tree method as compared to the linear hash chain method.
- During decryption, time cost includes calculation of class key using user's master key and calculation of time granule parameters. Due to which its time complexity is $\log_2(te-tb+1)$ as compared to linear hash chain method with time complexity of $\log_2(te-tb)$ where [te,tb] is a valid time interval for access by a particular user.

B. *CLOUDMASK MODEL*

Model: [5] focuses on the security and privacy of Data as a service (DaaS) facility of cloud. The CloudMask is an approach for securing EHR data as well. The entire data is divided and managed in the form of documents. The major entities involved in this system are:-

- Document Manager: It manages the subscription and encryption of documents.
- Cloud Service Provider: It provides the cloud service wherein the encrypted data is to be stored.
- Users: They could be anybody ranging from a cashier or healthcare staff to patient or a doctor. The data can be accessed and stored by them
- Identity Providers: They compute and provide unique identity tokens to users during data access.

The most significant part of this model is the usage of two protocols which are the Oblivious Commitment Based Envelope (OCBE) and Broadcast Group Key Management (BGKM). Firstly, the Oblivious Commitment Based Envelope (OCBE) Protocol is used by the Document Manager (DM), who manages subscriptions and encrypts data according to the access control policies. The user sends the DM an identity attribute according to a condition. The DM returns an envelope. The user can decrypt it only if he/she knows the attribute value committed. The DM will not come to know of this attribute. Secondly, the Broadcast Group Key Management Scheme is followed. In this, communication with a group of users is made secure by broadcasting the message to those users which they can decrypt by a symmetric key. But if the group dynamics change, if a person leaves or joins the group, new keys are not reissued. Each user is given a secret which can be combined with some public information to obtain the group key. Hence, a change in group dynamics leads to change of public information.

Implementation: The implementation of CloudMask is done in different phases. These phases are:

1. Token Issuance: A new user sends its identity attributes and its proof to the Identity Provider (IdP). The IdP verifies the authenticity of the proof and issues tokens to the user. Each token is unique and is given in a standard format of <ps, idt, c, ds> wherein ps is the pseudonym which uniquely identifies a user, idt is the identity attribute being considered, c is the Pedersen's commitment and ds is the digital signature given by the IdP.
2. Token Registration: Users need to register with the Cloud Service Provider before gaining access to the content. As mentioned above, the data is divided and subdivided into documents and subdocuments, respectively. A user, first, retrieves the access control policies which are of the form <subj, pid> where pid is the policy id and subj is the subject of the policy which is nothing but a set of conditions. These conditions must be satisfied to satisfy the policy. A condition could be any comparison statement where a particular value is compared to the identity attribute. Every subdocument has a policy configuration associated with it. In order to access the subdocument at least one of the conditions in the policy must be satisfied. The user provides its identity token to the Document Manager. Here, the OCBE protocols come into the picture. Every time a token is sent to the Document Manager, Conditional Subscription Secrets are issued by the Document Manager and are sent to the user. It contains the keys which allow users to decrypt the subdocument whose policies are satiated. This is done using the BGKM scheme
3. Publication and Authorization: The data is encrypted by the Document Manager and published. The subdocuments contain the unique identifier for each subdocument, encrypted data, encrypted metadata identifying keywords from actual data, hash–based message authentication code (HMAC) and access control vector. The users are given access to the metadata and data by using an authorization method i.e. users calculate their HMAC using their Conditional Subscription Secrets and public access control vector. If this HMAC matches the HMAC of the subdocument then access is given.

These phases provide access to the users but from a technical point of view, key management is an important task. The algorithmic procedures followed for key management are:

1. Setup(): The Document Manager initializes all the necessary parameters:
2. q→ an l-bit prime number,
3. N→ the size of group of users which is mostly set to n,
4. H→ a cryptographic hash function which generates F, which is a finite field with q elements,
5. KS→ key space which set to F,
6. SS→ Secret space which is l-bit random secrets, and
7. S→ used secrets which is set to null initially

8. SecGen(): The unused secret, s, in SS is picked at random for a particular user such that s is not an element of S.

9. KeyGen(S): The group key, k is picked up from KS, and n random bit strings are chosen from which are of l bits. A matrix A which is the Access Control Matrix, is formed in which each element $a_{i,j}$ is stored in the following form: if j=0, $a_{i,j}$ =1 else $a_{i,j}$ =H(s||z). The Document Manager chooses nonzero vector Y from the null space of A such that AY=0. Thus, the Access Control Vector(ACV) is computed where ACV=k.e+Y. e=(1,0,0,..,0) is the base vector for F. Now, private k and public PI is output where PI =(ACV,<z1,..,zn>).

10. KeyDer(s,PI): Using the PI and s, the user computes the Key Extraction Vector (KEV). This vector is a unique representation of a row of the matrix A. Thereby the group key k' is calculated by taking the inner product of the KEV and ACV.

11. Update(S): The key generation algorithm is run again to generate a new public information tuple. This is when the group dynamics change. By changing the PI only, new group key can be issued.

Advantages: The pros of the system include-

- Identity tokens are issued to the users based on their attributes but the Document Manager and Cloud Service Provider do not come to know of the attributes and privacy is maintained.
- The data and metadata is stored in the encrypted form due to which the Service Provider never learns of the data stored.
- Attribute based access control makes registration and authorization more secure.
- Cloud Service Provider charges users on the basis of bandwidth utilization. Hence, the HMAC authorization principle used in the system controls the bandwidth used by the user in the cloud.
- The system can be used with subset-cover techniques wherein each user is provided with multiple secrets, some of them may or may not be unique. When minimal secrets are chosen which cover all the members of the group, it makes the complexity sublinear with n and number of secrets needed is reduced to log n. This in turn improves the algorithms used in the system.

C. *TWO LEVEL ACCESS CONTROL MODEL*

Model: Usually, access control models propose solutions to enhance data sharing where read access control is stressed upon but [6] also, provides a solution to write access control. The roles participating in the system are:-

- Data owner: who owns the data, stores it in the cloud in the encrypted form and decides who gets what kind of privilege with the data
- Data user: who is allowed by the data owner to use the data and has a certain privilege over the data
- Cloud Provider: who stores the data in its service and upon verifying the access request may or may not provide access to a person other than the data owner

Data resources or files stored in the cloud are divided into access blocks. The two levels of access control utilized in the model are coarse-grained and fine-grained access control. The coarse-grained level view is provided to the cloud provider. These access blocks have privilege levels associated with them. When a request is made for the block, the cloud provider matches the request to the block itself. The fine-grained level view is for the users of the data. The owner provides access to the users on their request based on each file within a block. The cloud is unaware of these policies and inputs.

Implementation: The model is implemented in the form of two separate solutions, one for read access and one for write access. Thereafter, an integrated approach is suggested.

1. Read Access Control:

2. System Setup: At the fine-grained level, files are distributed into access blocks. For each block, generate a tree graph by running Publish(r; o; eo; acl) for each resource r owned by o with an initial set of ACLs. Encrypt resources using keys from internal nodes in the tree. At the coarse-grained level, compute parameters for a predicate encryption scheme. Each owner construct a separate tree graph over all resources he owns to distribute authorization tokens SKf =GenKeySK(f) based on the initial ACLs.

3. Access Authorization: At the fine-grained level, add a leaf node containing the new user's public key to the corresponding tree graph with encryption keys. Update the graph by adding new internal nodes and appropriate edges if necessary. Update file encryptions if new internal nodes were added previously. At the coarse-grained level, perform similar operations with respect to the tree graph containing read access tokens.

4. Access Request: At the fine-grained level, an authorized user u derives the decryption key from tree graph for resource r by calling Find Chain(u; r), Find Resources (u; r) and Compute Key(u; chain). At the coarse-grained level, he calls the same set of functions but to query the tree graph with access tokens and token cid = EncSK(id) for the requested _le id, and then submit a randomized token tid = Rand(cid) to the cloud.

5. Access Check: At the _ne-grained level, only authorized users can derive the correct decryption key for each _le using the public tree structure. At the coarse-grained level, the cloud provider executes File Access Check to identify the block that contains the requested file.

6. Access Rule Update: At the _ne-grained level, changes are applied immediately upon policy updates. If the policy update involves access revocation, the data owner changes the encryption of corresponding _les. The data owner identifies the blocks affected by those files and updates their tree graphs with decryption keys. The changes at the coarse-grained level happen at longer intervals of time, the length of which would depend on the resources of the data owner. They involve updating of the tree graph with access tokens.

7. Write Access Control:

- File Encryption: We apply an asymmetric encryption scheme to handle all possible combinations of read and write access to a file. Since such scheme is computationally expensive for large size of data, file content is still encrypted using a symmetric key (e.g., AES), which is further encrypted under the public key. Two trees are constructed for key distribution per block-one for the public(encryption) keys and the other for the private (decryption) keys. These two trees share the same set of internal nodes for a one to one correspondence between public and private key pair. Only files readable and writable by the same set of users can share the same public key pair.
- Access Authorization: Tokens Two trees are constructed by each data owner for the distribution of read and write access tokens respectively.
- File Identifiers for Write Updates: We observe that the write authorization token is a valid encryption for a predicate encryption that provides polynomials evaluation, and the structure of the encrypted plaintext for access to file id is a vector of the form (1; id; id2; : : : ; idn), where n is the number of files placed in a block. The structure of the ciphertext allows it to be split into parts where one part is an encryption of the vector (1; id; id2; : : : ; idk) (k < n, n > 2), which is no longer a valid write access token for that file, but can still be used identify file updates for users with read privilege. This can be achieved using a decryption predicate for a polynomial of degree k that has id as a zero point.

8. Integrated Read-Write Access Control:

- Setup: At the fine-grained level, construct a key distribution tree per block based on read access rules. For each node in the tree, generate a public-private key pair (skn; pkn), but only store the secret key skn. Construct another tree with the same set of nodes to store the public key pkn, with edges determined by write access rule. For each file id generate a AES key skaesid for encryption, and append to the ciphertext Encpkn(skaesid). At the coarse-grained level, each data owner generates a tree graph, where each node contains read access token Encpk'ra (id) and SKx-id = GenKeysk''-ra(f) where f(x) = x-id using predicate encryption with different keys. Similarly, construct another tree to distribute write access tokens Encpkwa (id).
- Access Authorization: At the coarse-grained level, extend the trees with read and write access tokens with new leaves for the new user and update the edges according to his read and write permissions. This may involve splitting of nodes and re-encrypting files with new keys if the user has read access only to a subset of files that have been encrypted with the same key.
- Write Access Request: At the fine-grained level, obtain the encryption key pkn for the file to be updated from the write tree. Encrypt the new content for that file with key pkn. At the coarse-grained level, submit to the cloud a re-randomized copy of the write authorization token for that file.
- Write Access Check: At the fine-grained level, a user can modify a file only if he has the encryption key and the write authorization token. Upon read he will check at the end of a block a list of updates with valid write access tokens. At the coarse-grained level, the cloud finds if there is a block for which the authorization token grants write access. The write access token is of the form (C0; fC1;i;C2;ign i=1), and the cloud uses the first components (C0; fC1;i;C2;ig2i=1) as an identifier for updates appended to a block.
- Write Access Rule Update: Update per-block trees for encryption keys and the tree for distributing write access tokens accordingly.

Advantages: Upon analysis of the various possible attacks on the system, the following results were found:

- For the privacy of the data owners, the cloud provider does not learn any of the content of the files that he stores. The cloud learns the frequency of access to particular blocks but not the exact files that have been accessed within a block. For users' privacy, the cloud provider cannot relate access requests to particular users', neither can he infer which requests were submitted from the same user. However, he can observe the block access pattern from the requests of all users. The data owner does not learn anything about the access requests for the data.
- For privacy of the data owners, the cloud provider learns how often update requests are submitted for each block but without finding out which files have been written. Similarly to the read requests, write requests coming from the users are anonymous and unlinkable. Thus the cloud provider cannot learn anything about the access behavior of a particular user, but only a cumulative view over the requests from all users.

D. *HSAC MODEL*

Model: In order to achieve the desired functionality described above, [7] proposes to make anonymous and oblivious access control decisions and grant access to resources based on tickets, using a universal homomorphic computation container (UHCC). The UHCC will allow for the integration of a suitable AC mechanism, such as a simple AC list, RBAC, or many other existing methods. We combine this to create the foundation for Homomorphic cryptography Supported Access Control (HSAC). Besides few assumptions about the infrastructure, this foundation allows the system to be tailored to the requirements of its application while achieving a maximum of privacy. We assume that the resource provider and the subject follow the push sequence for getting access to a resource. A subject obtains an authorisation ticket from an organisation's AAA Server (the PDP) and presents this ticket to the service equipment (the PEP), therefore proving that access was allowed by the PDP. This indirection allows for maximised privacy during the PDP's decision process. Using the push sequence, the PEP cannot know when or by whom the ticket was obtained. Compared to the more direct pull sequence (the PEP obtains the authorisation from PDP itself), the push sequence allows for the separation of the PDP and PEP and only increases communication complexity from one to a second exchange. This is desirable if the subject has an interest in maintaining his privacy and therefore in protecting the ticket and has the advantage of separation between PDP and PEP. However, PDP and PEP will usually reside within the same organisation, and hence an honest but curious provider can correlate information from both entities.

Implementation:-

- Homomorphic Container: The UHCC is a complete computer which takes in inputs for a program and produces outputs but all of these are in the encrypted form. The owner assembles a program which is encrypted with public key of the owner, present within a memory image. This image is sent to an executing entity. The parts of program which are modified due to outcomes are sent back to the owner. These encrypted outcomes are thereafter decrypted by the owner using its private key.

- HSAC Model can be implemented using an arbitrary function f. This function considers the subject credentials, policies and static environment variables. It takes Access Control decisions based on subject to role, role to permission and resource to permission mapping. Function f is computed and hidden from the provider. f, i.e. subject credentials and resource identifiers are encrypted using the public key of the subject and are wrapped into a homomorphic container. This is sent to the PDP. The encrypted function f is provided with inputs using subject's public key for compatibility. If outcome generated is 1, and the mapping is valid then a PDP issues a ticket which is encrypted using the PDP's private key. This ticket is sent to the subject. The subject decrypts the container using its private key. The ticket sent is verified by the subject and PEP using the PDP's public key. If the ticket is valid PEP gives permission to the subject to access the resource.

Advantages: After considering attacks from malicious adversaries apart from network attacks, the model offers the following advantages:

- The request parameters travelling from subject to provider are encrypted which prevents getting hold of this information by the provider, i.e. information regarding the subject credentials

- The ticket issued by the PDP cannot be modified either as the PDP digitally signs the ticket and it can be accessed using the PDP's public key only

- The outcome generated as a result of execution of the program at PDP, is injected into predetermined memory locations of the image and only those parts are sent due to which the plaintext or the PDP's private key cannot be acquired.

E. PRIVACY AWARE ACCESS CONTROL MODEL

Model: [8] proposes a privacy aware access control system that provides two levels of protection for user's data stored on a CSP. The CSP is responsible for protecting user's data from unauthorized users, while user's data is protected from the CSP with the help of a third-party service provider. Instead of using complicated PKI paradigm, we employ multiple layers of commutative encryption are used along with a third party service provider to protect the users' data from the CSP while there is no need for users to trust the third party service provider either. Our approach separates the data protection against CSPs from the data protection against unauthorized users. We use commutative encryption to protect data against CSPs while an authorization mechanism enforced by the CSP is responsible for data protection against unauthorized users. By this separation, our proposed approach enables data owners to protect their resources from untrusted CSPs using encryption while allowing them to share their resources with authorized users and protect their resources from unauthorized users using fine-grained access control policies. The granularity level of these policies depends on what access control mechanism the CSP supports.

The major entities which are considered to participate in this system are:

- the data owner who creates the data and store it on the cloud service provider in an encrypted format and determines who has access to the data
- the data consumer who may have access to the protected data depending on the access control policies defined by the data owner
- the cloud service provider that stores the encrypted data and responds to access requests by the data

Implementation:-

- Setup Phase: The setup process is executed only once during the installation phase and its goal is to initialize required parameters for cryptographic operations on all parties, the CSP, the PMS, and the user. It determines a symmetric key space, a symmetric key initialization vector space, a symmetric stream cipher, a cryptographically secure hash function, a HMAC-based key derivation function, and a random master secret.
- Data Encryption: Before moving the data to the CSP, the data owner encrypts the data using a (non-commutative) symmetric key k. Then, the resulting cipher text is transferred and stored on the CSP. The data owner defines access control policies which are enforced by access control mechanism of the CSP on the encrypted data.
- Key Encryption Algorithm: Once the data owner encrypts the data using the key k and stores it on the CSP, we need to share this key with authorized data consumers, so they (and only they) can decrypt the data. Since neither of the CSP nor the PMS are trusted, they should not be able to access the key k. We use multiple layers of commutative encryption to encrypt and share the key k with authorized data consumers. The key encryption algorithm 1 used for this purpose includes the following procedures.
  Add-DataOwner-CELayer: This procedure is performed on the data owner side before sharing the key. Its input is the clear text key clrtxtkeyo$\in$ $\{0, 1\}^*$and the steps 1-6 of the algorithm 1 are executed.

Add-CSP-CELayer: After the data owner successfully runs the Add-DataOwner-CELayer procedure, the CSP runs the Add-CSP-CELayer procedure. Its input is the cipher text key cphtxtkyo and the CSP runs the steps 7-11 of the algorithm 1 to add another layer to the cipher text key cphtxtkyo. Remove-DataOwner-CELayer: This procedure is run on the data owner side after receiving the cipher text key from the CSP. Its inputs are _ivcs, cphtxtkeyocs_ from the Add-CSP-CELayer procedure as well as kstro and Digo from Add-DataOwner-CELayer procedure. This procedure removes the data owner's encryption layer from the cipher text key cphtxtkeyocs by running the steps 12-13 of the algorithm1. Algorithm 1 Key Encryption Algorithm Input: clrtxtkeyo This part is performed at the data owner side (Add-DataOwner-CELayer)

```
1: Generate ivo ∈ IVu
2: Generate cko ∈ CKu
3: Digo ← Hu(clrtxtkeyo)
4: kstro ← SymStru(ivo, cko)
5: cphtxtkeyo ← clrtxtkeyo ⊕ kstro
6: Send _cphtxtkeyo_ to the CSP
This part is performed at the CSP side (Add-CSPCELayer)
7: ckcs ← HKDFcs(hn,Xcs)
8: Generate ivcs ∈ IVcs
9: kstrcs ← SymStrcs(ivcs, ckcs)
10: cphtxtkeyocs ← cphtxtkeyo ⊕ kstrcs
11: Send _ivcs, cphtxtkeyocs_ to the data owner
This part is performed at the data owner side (Remove-DataOwner-CELayer)
12: cphtxtkeycs ← cphtxtkeyocs ⊕ kstro
13: Send _cphtxtkeycs, ivcs,Digo_ to the PMS
```

- Key Decryption Algorithm: Once the data consumer is authorized and granted access to the cipher text data according to the access control policies, she receives the key k that was used to encrypt that data. The key k itself is encrypted using the CSP's encryption layer and the data consumer should be able to decrypt the key. The key decryption algorithm 2 is used by the data consumer to recover the clear text key. It includes the following procedures.

  Add-DataConsumer-CELayer: This procedure is performed locally at the data consumer side. The input is the tuple _cphtxtkeycs, ivcs,Digo_ and the steps 1-5 of the algorithm 2 are executed.

  Remove-CSP-CELayer: After a data consumer successfully runs the Add-DataConsumer-CELayer procedure, the CSP runs the Remove-CSP-CELayer procedure. Its inputs are the cipher text cphtxtkeycsc and the initialization vector ivcs. The steps 6-9 of the algorithm 2 are performed by the CS to remove its encryption layer from cphtxtkeycsc.

  Remove-DataConsumer-CELayer: After the Add-DataConsumer-CELayer and the Remove-CSP-CELayer procedures were run successfully, the data consumer runs the Remove-DataConsumer-CELayer procedure to recover the clear text key. Its inputs are the result from the Remove-CSP-CELayer procedure as well as ivc , kstrc and Digo from the Add-DataConsumer-CELayer procedure. As shown in steps 10-12 of the algorithm 2, it decodes cphtxtkeyc with kstrc and calculates the digest of the result using Hu. If the digest equals to Digo, it accepts the clear text key, otherwise the key is rejected because its integrity has been violated. If the data has not been manipulated by potential adversaries, clrtxtkeyc should be same as clrtxtkeyo.

Algorithm 2 Key Decryption Algorithm
Input: cphtxtkeycs and ivcs and Digo

```
This part is performed at the data consumer side (Add-DataConsumer-CELayer)
1: Generate ivc ∈ IVu
2: Generate ckc ∈ CKu
3: kstrc ← SymStru(ivc, ckc)
4: cphtxtkeycsc ← cphtxtkeycs ⊕ kstrc
5: Send _cphtxtkeycsc, ivcs_ to the CSP
This part is performed at the CSP side (Remove-CSP-CELayer)
6: ckcs ← HKDFcs(Hcs(c),Xcs)
7: kstrcs ← SymStrcs(ivcs, ckcs)
8: cphtxtkeyc ← cphtxtkeycsc ⊕ kstrcs
9: Send cphtxtkeyc to the data consumer
This part is performed at the data consumer side (Remove-DataConsumer-CELayer)
10: clrtxtkeyc ← cphtxtkeyc ⊕ kstrc
11: Digc ← Hu(clrtxtkeyc)
12: Compare Digc and Digo and accept the clear text key if they are equal.
```

Advantages: The proposed approach offers the following advantages.

- It does not require any public key infrastructure or key distribution scheme.
- If access control policies change, there is no need to re-encrypt the data or re-generate and re-distribute encryption keys. Since our approach separates the data protection against CSPs from the data protection against unauthorized users, changes in policies which are used for data protection against unauthorized users do not affect the encryption part of the approach which is used for data protection against the CSP.
- It is possible to enable data owners to define policies at various granularity levels such as role-based, attribute based, group based, etc.
- As the encryption part is independent from CSP's access control mechanism, data owners are able to protect data from mistrusted CSPs regardless of access control mechanism the CSPs support and types of access policies they can define.
- It provides flexibility in terms of types of access control policies that the data owner can define; various types of access control policies can be enforced.
- The enforcement of access control policies that protect data from unauthorized users is performed by the CSP and the encryption part is shared between the CSP, the PMS and users. So, data owner does not have to be in charge of huge part of enforcement while being able to protect his resources from mistrusted CSPs.

- It is scalable since the data encryption part is separated from access control policies and no public key infrastructure, key distribution scheme or re-encryption is needed.

Unauthorized User: If an unauthorized user is able to break the CSP's access control mechanism somehow and get access to the cipher text, he cannot decrypt the cipher text as the key which was used for encryption is stored on the PMS. On the other hand, if an unauthorized user is able to break the PMS and get access to the cipher text key and decipher it, he cannot do anything as he does not have access to the cipher text which is stored on the CSP. An unauthorized user is able to break the system if and only if he can break both the CSP's access control system and the PMS at the same time. In this case, he would be able to decrypt the key using a similar process as data consumer uses in our proposed approach (cf. key decryption algorithm) and consequently decrypt the data. However, if the access control policies are properly enforced by the CSP unauthorized users will not able to get access to the cipher text stored on the CSP.

PMS Provider: When the key is stored on the PMS provider, it is protected by an encryption layer from CSP. During encryption phase, the data owner encrypts the clear text key clrtxtkeyo using key stream kstro which is in turn encrypted using the CSP's encryption layer and only the final result cphtxtkeycs $\leftarrow$ cphtxtkeyocs $\oplus$ kstro is sent to the PMS provider. The key is encrypted using the CSP's encryption layer and the PMS cannot extract the clear text key.

Cloud Service Provider: The data stored on the CSP is encrypted using the data owner's key. In order to decrypt the encrypted data, the CSP must get access to the key k which was used for encryption. When the key is sent to the CSP, it is protected by an encryption layer from the data owner. During decryption phase, the data consumer first adds an encryption layer to the already encrypted key cphtxtkeycs using key stream kstrc and sends the result cphtxtkeycsc = cphtxtkeycs $\oplus$ kstrc to the CSP. In both cases, the key is encrypted and the CSP cannot extract the clear text key.

Bit-Flip Attack It has been shown that XOR-based stream ciphers are susceptible to bit-flip attacks. Stream ciphers usually encrypt and decrypt data, one bit at a time by XORing the plain text with a key stream. Because of this, an attacker could modify one bit of cipher text through a man in the middle attack or replay attack without knowing the key and the recipient of the cipher text would not not know the data had changed. In order to mitigate this attack and provide message integrity, while adding an encryption layer to the key k in the Add-DataOwner-CELayer procedure, we calculate a message digest at the data owner side and send it along with the encrypted key to the PMS while running the Remove- DataOwner-CELayer procedure. The recipient of the key verifies the message digest in the Remove-DataConsumer-CELayer procedure and rejects the key if the digest of the recovered clear text key is not correct.

Key Reuse Attack: If the same key is used for two or more different messages, XOR-based encryption mechanism is susceptible to key reuse attacks. The attacker can eliminate the encryption key by applying XOR to the encrypted messages by themselves. We use AES in countermode to avoid the key reuse attack. The key stream is derived from a secret key and a random initialization vector (IV). The IV can be sent in the clear and combined with a secret master key, it can be used to create a one-time key for the stream cipher. The AES in counter mode (CTR) is not susceptible to key-reuse attack, if the same combination of secret key and IV is not used more than once to generate a key stream.

Sr.No.	Paper	Model	Publication Year	Advantages	Defects	Resources
1.	RBTBAC: Secure Access and Management of EHR Data	Role based Time Bound Access Control Model	June, 2011	User privacy maintained, digital signature usage prevents unauthorized attacks, credential revocation list prevents usage of expired credentials	Key distribution for different classes makes key handling difficult	3rd International Workshop on e-Healthcare Information Security
2.	Towards Privacy Preserving Access Control in the Cloud	CloudMask Model	October, 2011	Group key generation and key handling as well as rekeying approach is user-friendly	Not practical for a large number of people within a group, change of policies leads to re-encryption	International Conference on Collaborative Computing: Networking, Applications and Work sharing
3.	Privacy Enhanced Access Control for Outsourced Data sharing	Two Level Access Control Model	March, 2012	Solutions for read and write access control policies is satisfied, use of two levels of access control protects data from cloud provider	Grouping of files into access blocks is difficult, if not done properly may lead to inefficiency	Financial Cryptography and Data Security
4.	Towards Privacy-Preserving Access Control with Hidden Policies, Hidden Credentials and Hidden Decisions	Homomorphic cryptography Supported Access Control Model	July, 2012	Policies and credentials defined by user are hidden, anonymity of user is maintained	Computations within the container take a lot of time, anonymity of user makes it difficult to maintain log information	Tenth Annual International Conference on Privacy, Security and Trust, IEEE
5.	Privacy Aware Access Control for Data Sharing in Cloud Computing	Privacy Aware Access Control Model	June, 2014	Flexible access control policy types, scalable as encryption and access control policy part is different, two levels of protection of data, commutative encryption employed and is reliable and easy	Data hidden but policies defined by the entities is not hidden	SCC'14, ACM

TABLE I. Comparison of various access control and privacy preservation models

CONCLUSION

Security of cloud is a major limitation which disables users from completely utilizing the cloud platform. The privacy preservation and access control of the cloud, if made secure enough will give users the freedom to store much more confidential data and make cloud data storage more popular. Hence, numerous approaches have been devised to solve the problem. The shortcomings of the considered approaches gave rise to a privacy aware access control system.

Thus, the proposed privacy aware access control system for data sharing applies two levels of encryption as well as offers simpler key management technique

REFERENCES

[1] Cloud Computing: Theory and Practice, Dan C. Marinescu, ACM, 2012

[2] A Survey of Risks, Threats and Vulnerabilities in Cloud Computing, Kamal Dahbur, Bassil Mohammad, Ahmad Bisher Tarakji, ACM 2011.

[3] R. Chow, P. Golle, M. Jackobsson, E. Shi, J. Staddon, R. Masouka, and J. Mollina. "Controlling data on the cloud: outsourcing computations without outsourcing control." Proc. Cloud Computing Security Workshop (CCSW09), pp. 85-90, 2009.

[4] R. Zhang, L. Liu, J. Li and Z. Han, "RBTBAC: Secure Access and Management of EHR Data," In Proc. Of the 3rd International Workshop on e-Healthcare Information Security (e-HISec 2011), June 27-29, 2011.

[5] M. Nabeel, E. Bertino, B. Thuraisingham, and M. Kantarcioglu, "Towards Privacy Preserving Access Control in the Cloud," In Proc. of the International Conference on Collaborative Computing: Networking, Aplications and Worksharing (CollaborateCom), pp.172-180, Orlando, Florida, USA, October 15-18, 2011.

[6] M. Raykova, H. Zhao, and S. M. Bellovin, "Privacy Enhanced Access Control for Outsourced Data sharing," In Proc. of the Financial Cryptography and Data Security, March 2012.

[7] Towards Privacy-Preserving Access Control with Hidden Policies, Hidden Credentials and Hidden Decisions, Marian Harbach, Sascha Fahl, Michael Brenner, Thomas Muders and Matthew Smith, 2012 Tenth Annual International Conference on Privacy, Security and Trust, IEEE.

[8] Privacy Aware Access Control for Data Sharing in Cloud Computing Environments Hassan Takabi, ACM June,2014

[9] D. Wei, "Commutative-like Encryption: A New Characterization of ElGamal," The Computing Research Repository, vol. 1011, 2010

[10] RSA Laboratories, "Public-Key Cryptography Standards (PKCS)," http://www.rsa.com/rsalabs/node.asp?id=2124

[11] Y. Zhu, H.X. Hu, G.J. Ahn, H.X. Wang, and S.B. Wang, "Provably Secure Role-Based Encryption with Revocation Mechanism", Journal of Computer Science and Technology, Vol. 26, No. 4, pp. 697--710, 2011.

a.