Secured Access In Cloud Computing

Ganesa Murthy. M *, Venkatesh. S **
Ganesa murthy.M*, student, IT, VelTech High Tech Dr.RR. Dr.SR. Engineering College, Chennai, India.

Venkatesh.S**, student, CSE, Sree Sastha College of Engineering, Chennai, India.

***Abstract*— Cloud Computing offers the space to share distributed resources and services that belong to different organizations or sites. Since Cloud computing share distributed resources via network over the open environment it makes security problems and also causes data conflict while storing. Since the data transmission on the internet or over any networks are vulnerable to the hackers attack. We are in great need of encrypting the data. To build a trusted computing environment for Cloud Computing system by providing Secure cross platform in to Cloud computing system. In this method some important security services including authentication, encryption and decryption and compression are provided in Cloud computing system.**

Keywords- Security, Data conflict, Data Encryption, Compression, Data Decryption .

IJERT

I. INTRODUCTION

Cloud computing is a promising computing paradigm which enables efficient data storage by its service structure. By fusing a set of techniques from research areas such as Service-Oriented Architectures (SOA) and virtualization, cloud computing is regarded as such a computing paradigm in which resources in the computing infrastructure are provided as services over the Internet, which can be described by terminology of “X as a service (XaaS)” where X could be software, hardware, data storage, and etc. which provide users with scalable resources in the pay-as-you use fashion at relatively low prices. As compared to building their own infrastructures, users are able to save their investments significantly by migrating businesses into the cloud. With the increasing development of cloud computing technologies, it is not hard to imagine that in the near future more and more businesses will be moved into the cloud.

As the logical expression can represent any desired data file set, fine-grainedness of data access control is achieved. To enforce these access structures, we define a public key component for each attribute. Data files are encrypted using public key components corresponding to their attributes. User secret keys are defined to reflect their access structures so that a user is able to decrypt a cipher text if and only if the data file attributes satisfy his access structureOne extremely challenging issue with this design is the implementation of user revocation, which would inevitably require re-encryption of data files accessible to the leaving user, and may need update of secret keys for all the remaining users. If all these tasks are performed by the data owner himself/herself, it would introduce a heavy computation overhead on him/her and may also require the data owner to be always online. To resolve this challenging issue, our proposed scheme enables the data owner to delegate tasks of data file re-encryption and user secret key update to cloud servers without disclosing data contents or user access privilege information. We achieve our design goals by exploiting a novel cryptographic primitive, namely key policy attribute-based encryption.

II. MODELS AND ASSUMPTIONS

First, confirm that we have the correct idea by comparing our modules and assumptions,

A. System Models

Similar to Enabling Public Verifiability and Data Dynamics for Storage Security in Cloud Computing, we assume that the system is composed of the following parties: the Data Owner, many Data Consumers, many Cloud Servers, and a Third Party Auditor if necessary. To access data files shared by the data owner, Data Consumers, or users for brevity, download data files of their interest from Cloud Servers and then decrypt. Neither the data owner nor users will be always online. They come online just on the necessity basis. For simplicity, we assume that the only access privilege for users is data file reading. Extending our proposed scheme to support data file writing is trivial by asking the data writer to sign the new data file on each update as does. From now on, we will also call data files by files for brevity. Cloud Servers are always online and operated by the Cloud Service Provider (CSP). They are assumed to have abundant storage capacity and computation power. The Third Party Auditor is also an online party which is used for auditing every file access event. In addition, we also assume that the data owner can not only store data files but also run his own code on Cloud Servers to manage his data files. This assumption coincides with the unified ontology of cloud computing which is proposed by Youseff et al meet the conscience of our proposal.

B. *Security Models*

In this work, we just consider Honest but Curious Cloud Servers as Over-encryption: Management of access control evolution on outsourced data [14] does. That is to say, Cloud Servers will follow our proposed protocol in general, but try to find out as much secret information as possible based on their inputs. More specifically, we assume Cloud Servers are more interested in file contents and user access privilege information than other secret information. Cloud Servers might collude with a small number of malicious users for the purpose of harvesting file contents when it is highly beneficial. Communication channel between the data owner/users and Cloud Servers are assumed to be secured under existing security protocols such as SSL. Users would try to access files either within or outside the scope of their access privileges. To achieve this goal, unauthorized users may work independently or cooperatively. In addition, each party is preloaded with a public/private key pair and the public key can be easily obtained by other parties when necessary.

C. *Design Goals*

Our main design goal is to help the data owner achieve fine-grained access control on files stored by Cloud Servers. Specifically, we want to enable the data owner to enforce a unique access structure on each user, which precisely designates the set of files that the user is allowed to access. We also want to prevent Cloud Servers from being able to learn both the data file contents and user access privilege information. In addition, the proposed scheme should be able to achieve security goals like user accountability and support basic operations such as user grant/revocation as a general one-to-many communication system would require. All these design goals should be achieved efficiently in the sense that the system is scalable.

III. SYSTEM STUDY

Before we begin your paper, first complete study of existing and enhancement in this past systems are repeatedy observed they are,

A. *Existing system*

Our existing solution applies cryptographic methods by disclosing data decryption keys only to authorized users. These solutions inevitably introduce a heavy computation overhead on the data owner for key distribution and data management when fine grained data access control is desired, and thus do not scale well.

a) *Disadvantages*

- **Software update/patches -** could change security settings, assigning privileges too low, or even more alarmingly too high allowing access to your data by other parties.
- **Security concerns -** Experts claim that their clouds are 100% secure - but it will not be their head on the block when things go awry. It's often stated that cloud computing security is better than most enterprises. Also, how do you decide which data to handle in the cloud and which to keep to internal systems - once decided keeping it secure could well be a full-time task?
- **Control -** Control of your data/system by third-party. Data - once in the cloud always in the cloud! Can you be sure that once you delete data from your cloud account will it not exist anymore or will traces remain in the cloud?

B. *Proposed System*

IJERT

a) *Main Idea*

In order to achieve secured access on outsourced data in the cloud, the proposed network consists of backup sites for recovery after disaster . the backup sites are located at remote location from the main server. If anyone of the paths fails it uses alternate path working. The encrypted file will be create during backup sites and data's are compressed. The data will be decrypted during recovery operation.
Then we use three operations:

1. Data backup operation

Client send the data to the server which is known as main server. At the same time data is also backup to multi servers.
In this method for data backup it involve with muti servers. such as (SA 1(Server, Application),SA 2,SA3,etc,…)

2. Data encryption and compression

we utilize and uniquely combine the following three advanced cryptographic techniques:

- Key Policy Attribute-Based Encryption (KP-ABE).
- Proxy Re-Encryption (PRE)
- Lazy re-encryption

Then for compression GZIP algorithm is used and for symmetric splitting of files SFSPL algorithm is used.

3. Data decryption and Decompression

This can be done with suitable decrypting techniques for the above techniques.

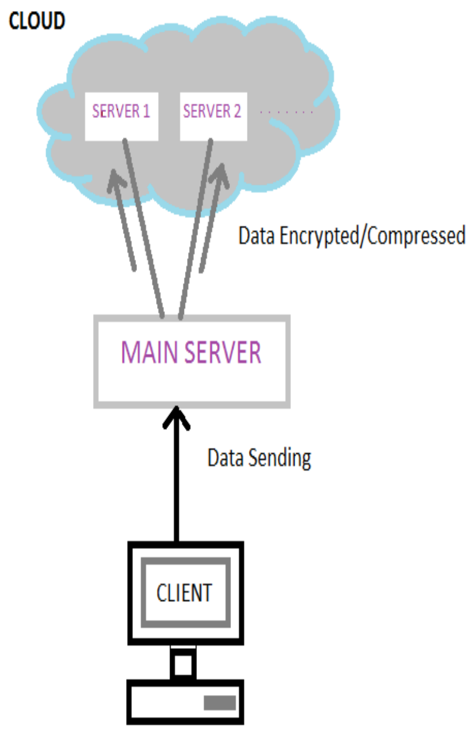

Fig.1: Data Backup

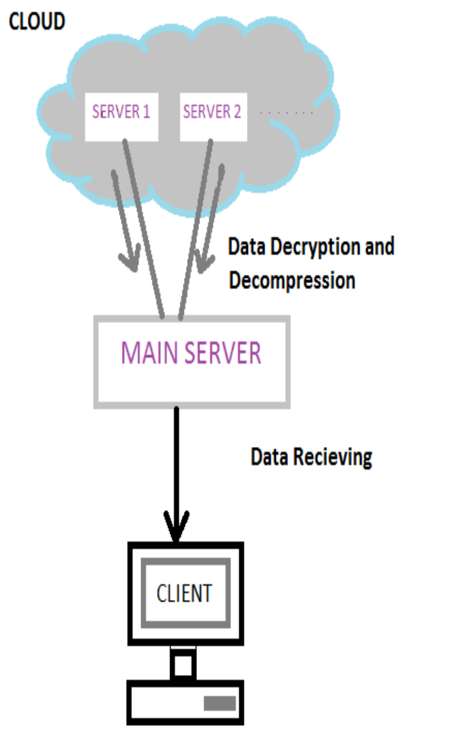

Fig. 2 : Data Recovery

More specifically, we associate each data file with a set of attributes, and assign each user an expressive access structure which is defined over these attributes. To enforce this kind of access control, we utilize KP-ABE to escort data encryption keys of data files. Such a construction enables us to immediately enjoy fine-grainedness of access control. However, this construction, if deployed alone, would introduce heavy computation overhead and cumbersome online burden towards the data owner, as he is in charge of all the operations of data/user management. Specifically, such an issue is mainly caused by the operation of user revocation, which inevitably requires the data owner to re-encrypt all the data files accessible to the leaving user, or even needs the data owner to stay online to update secret keys for users. To resolve this challenging issue and make the construction suitable for cloud computing, we uniquely combine PRE with KP-ABE and enable the data owner to delegate most of the computation intensive operations to Cloud Servers without disclosing the underlying file contents. Such a construction allows the data owner to control access of his data files with a minimal overhead in terms of computation effort and online time, and thus fits well into the cloud environment. Data confidentiality is also achieved since Cloud Servers are not able to learn the plaintext of any data file in our construction. For further reducing the computation overhead on Cloud Servers and thus saving the data owner's investment, we take advantage of the lazy re-encryption technique and allow Cloud Servers to "aggregate" computation tasks of multiple system operations. As we will discuss in section V-B, the computation complexity on Cloud Servers is either proportional to the number of system attributes, or linear to the size of the user access structure/tree, which is independent to the number of users in the system. Scalability is thus achieved. In addition, our construction also protects user access privilege information against Cloud Servers. Accountability of user secret key can also be achieved by using an enhanced scheme of KP-ABE.

IJERT

b) *Definition and Notation*

For each data file the owner assigns a set of meaningful attributes which are necessary for access control. Different data files can have a subset of attributes in common. Each attribute is associated with a version number for the purpose of attribute update as we will discuss later. Cloud Servers keep an attribute history list *AHL* which records the version evolution history of each attribute and PRE keys used. In addition to these meaningful attributes, we also define one dummy attribute, denoted by symbol Att_D for the purpose of key management. Att_D is required to be included in every data file's attribute set and will never be updated. The access structure of each user is implemented by an access tree. Interior nodes of the access tree are threshold gates. Leaf nodes of the access tree are associated with data file attributes. For the purpose of key management, we require the root node to be an *AND* gate (i.e., *n*-of-*n* threshold gate) with one child being the leaf node which is associated with the dummy attribute, and the other child node being any threshold gate. The dummy attribute will not be attached to any other node in the access tree. Fig.1 illustrates our definitions by an example. In addition, Cloud Servers also keep a user list *UL* which records *ID*s of all the valid users in the system. Fig.2 gives the description of notation to be used in our scheme.

Notation	**Description**
PK, MK	system public key and master key
T_i	public key component for attribute i
t_i	master key component for attribute i
SK	user secret key
sk_i	user secret key component for attribute i
E_i	cipher-text component for attribute i
I	attribute set assigned to a data file
DEK	symmetric data encryption key of a data file
P	user access structure
L_P	set of attributes attached to leaf nodes of P
Att_D	the dummy attribute
UL	the system user list
AHL_i	attribute history list for attribute i
$rk_{i \leftrightarrow i'}$	proxy re-encryption key for attribute i from its current version to the updated version i'
$\delta_{O,X}$	the data owner's signature on message X

Fig. 3: Notation used in our scheme description

c) *Scheme Description*

For clarity we will present our proposed scheme in two levels: *System Level* and *Algorithm Level*. At system level, we describe the implementation of high level operations, i.e., *System Setup*, *New File Creation*, *New User Grant*, and *User Revocation*, *File Access*, *File Deletion*, and the interaction between involved parties. At algorithm level, we focus on the implementation of low level algorithms that are invoked by system level operations.

1) System Level Operations: System level operations in our proposed scheme are designed as follows.

System Setup In this operation, the data owner chooses a security parameter κ and calls the algorithm level interface *ASetup*(k), which outputs the system public parameter *PK* and the system master key *MK*. The data owner then signs each component of *PK* and sends *PK* along with these signatures to Cloud Servers.

New File Creation Before uploading a file to Cloud Servers, the data owner processes the data file as follows.
- select a unique *ID* for this data file;
- randomly select a symmetric data encryption key $DEK \overset{R}{\leftarrow} K$, where K is the key space, and encrypt the data file using *DEK*;
- define a set of attribute I for the data file and encrypt *DEK* with I using KP-ABE,

i.e., $(\tilde{E}, \{E_i\}_{i \in I}) \leftarrow AEncrypt(I, DEK, PK)$.

Header

Body

ID	I, $\tilde{E}$, $\{E_i\}_{i \in i}$	$\{DataFile\}_{DEK}$

Fig. 4: Format of a data file stored on the cloud

IJERT

Finally, each data file is stored on the cloud in the format as is shown in Fig.3.

New User Grant When a new user wants to join the system, the data owner assigns an access structure and the corresponding secret key to this user as follows.
- assign the new user a unique identity w and an access structure P;
- generate a secret key SK for w, i.e., $SK \leftarrow AKeyGen(P,MK)$;
- encrypt the tuple $(P, SK, PK, \delta_{O,(P,SK,PK)})$ with user w's public key, denoting the cipher-text by C;
- send the tuple $(T, C, \delta_{O,(T,C)})$ to Cloud Servers, where T denotes the tuple $(w, \{j, sk_j\}_{j \in LP \setminus AttD})$. On receiving the tuple $(T, C, \delta_{O,(T,C)})$, Cloud Servers processes as follows.
  - verify $\delta_{O,(T,C)}$ and proceed if correct;
  - store T in the system user list UL;
  - forward C to the user.

On receiving C, the user first decrypts it with his private key. Then he verifies the signature $\delta_{O,(P,SK,PK)}$. If correct, he accepts (P, SK, PK) as his access structure, secret key, and the system public key.

As described above, Cloud Servers store all the secret key components of *SK* except for the one corresponding to the dummy attribute Att_D. Such a design allows Cloud Servers to update these secret key components during user revocation as we will describe soon. As there still exists one undisclosed secret key component (the one for Att_D), Cloud Servers cannot use these known ones to correctly decrypt ciphertexts. Actually, these disclosed secret key components, if given to any unauthorized user, do not give him any extra advantage in decryption as we will show in our security analysis.

User Revocation We start with the intuition of the user revocation operation as follows. Whenever there is a user to be revoked, the data owner first determines a minimal set of attributes without which the leaving user's access structure will never be satisfied. Next, he updates these attributes by redefining their corresponding system master key components in *MK*. Public key components of all these updated attributes in *PK* are redefined accordingly. Then, he updates user secret keys accordingly for all the users except for the one to be revoked. Finally, *DEK*s of affected data files are re-encrypted with the latest version of *PK*. The main issue with this intuitive scheme is that it would introduce a heavy computation overhead for the data owner to re-encrypt data files and might require the data owner to be always online to provide secret key update service for users. To resolve this issue, we combine the technique of proxy re-encryption with KP-ABE and delegate tasks of data file re-encryption and user secret key update to Cloud Servers. More specifically, we divide the user revocation scheme into two stages as is shown below.

In the first stage, the data owner determines the minimal set of attributes, redefines *MK* and *PK* for involved attributes, and generates the corresponding PRE keys. He then sends the user's *ID*, the minimal attribute set, the PRE keys, the updated public key components, along with his signatures on these components to Cloud Servers, and can go off-line again. Cloud Servers, on receiving this message from the data owner, remove the revoked user from the system user list *UL*, store the updated public key components as well as the owner's signatures on them, and record the PRE key of the latest version in the attribute history list *AHL* for each updated attribute. *AHL* of each attribute is a list used to record the version evolution history of this attribute as well as the PRE keys used. Every attribute has its own *AHL*. With *AHL*, Cloud Servers are able to compute a single PRE key that enables them to update the attribute from any historical version to the latest version. This property allows Cloud Servers to update user secret keys and data files in the "lazy" way as follows. Once a user revocation event occurs, Cloud Servers just record information submitted by the data owner as is previously discussed. If only there is a file data access request from a user, do Cloud Servers re-encrypt the requested files and update the requesting user's secret key. This statistically saves a lot of computation overhead since Cloud Servers are able to "aggregate" multiple update/re-encryption operations into one if there is no access request occurring across multiple successive user revocation events.

File Access This is also the second stage of user revocation. In this operation, Cloud Servers respond user request on data file access, and update user secret keys and re-encrypt requested data files if necessary. As is depicted in Fig. 4, Cloud Servers first verify if the requesting user is a valid system user in *UL*. If true, they update this user's secret key components to the latest version and re-encrypt the *DEK*s of requested data files using the latest version of *PK*. Notably; Cloud Servers will not perform update/re-encryption if secret key components/data files are already of the latest version. Finally, Cloud Servers send updated secret key components as well as ciphertexts of the requested data files to the user. On receiving the response from Cloud Servers, the user first verifies if the claimed version of each attribute is really newer than the current version he knows. For this purpose, he needs to verify the data owner's signatures on the attribute information (including the version information) and the corresponding public key components, i.e., tuples of the form (j, T'_j) in Fig. 4. If correct, the user further verifies if each secret key component returned by Cloud Servers is correctly computed. He verifies this by computing a bilinear pairing between sk'_j and T'_j and comparing the result with that between the old sk_j and T_j that he possesses. If verification succeeds, he replaces each sk_j of his secret key with sk'_j and update T_j with T'_j. Finally, he decrypts data files by first calling *ADecrypt*(*P*, *SK*,*E*) to decrypt *DEK*'s and then decrypting data files using *DEK*'s.

File Deletion This operation can only be performed at the request of the data owner. To delete a file, the data owner sends the file's unique *ID* along with his signature on this *ID* to Cloud Servers. If verification of the owner's signature returns true, Cloud Servers delete the data file. *2) Algorithm level operations:* Algorithm level operations include eight

IJERT

algorithms: *ASetup*, *AEncrypt*, *AKeyGen*, *ADecrypt*, *AUpdateAtt*, *AUpdateSK*, *AUpdateAtt4File*, and *AMinimalSet*. As the first four algorithms are just the same as *Setup*, *Encryption*, *Key Generation*, and *Decryption* of the standard KP-ABE respectively, we focus on our implementation of the last four algorithms.

AUpdateAtt This algorithm updates an attribute to a new version by redefining its system master key and public key component. It also outputs a proxy re-encryption key between the old version and the new version of the attribute.

AUpdateAtt4File This algorithm translates the ciphertext component of an attribute i of a file from an old version into the latest version. It first checks the attribute history list of this attribute and locates the position of the old version. Then it multiplies all the PRE keys between the old version and the latest version and obtains a single PRE key. Finally it apply this single PRE key to the ciphertext component E_i and returns $E^{(n)}_i$ which coincides with the latest definition of attribute i.

AUpdateSK This algorithm translates the secret key component of attribute i in the user secret key SK from an old version into the latest version. Its implementation is similar to *AUpdateAtt4File* except that, in the last step it applies $(rk_{i\leftrightarrow i(n)})^{-1}$ to SK_i instead of $rk_{i\leftrightarrow i(n)}$. This is because t_i is the denominator of the exponent part of SK_i while in E_i it is a numerator.

AMinimalSet This algorithm determines a minimal set of attributes without which an access tree will never be satisfied. For this purpose, it constructs the conjunctive normal form (CNF) of the access tree, and returns attributes in the shortest clause of the CNF formula as the minimal attribute set.

IV. DESCRIPTION

A. *Key Policy Attribute Based Encryption (KP-ABE)*

KP-ABE [15] is a public key cryptography primitive for one-to-many communications. In KP-ABE, data are associated with attributes for each of which a public key component is defined. The encryptor associates the set of attributes to the message by encrypting it with the corresponding public key components. Each user is assigned an access structure which is usually defined as an access tree over data attributes, i.e., interior nodes of the access tree are threshold gates and leaf nodes are associated with attributes. User secret key is defined to reflect the access structure so that the user is able to decrypt a cipher text if and only if the data attributes satisfy his access structure [23]. A KP-ABE scheme is composed of four algorithms which can be defined as follows:

- Setup Attributes
- Encryption
- Secret key generation
- Decryption

1) *Setup Attributes:*

This algorithm is used to set attributes for users. This is a randomized algorithm that takes no input other than the implicit security parameter. It defines a bilinear group G_1 of prime order p with a generator g, a bilinear map $e : G_1 \times G_1 \rightarrow G_2$ which has the properties of *bilinearity*, *computability*, and *non-degeneracy*. From these attributes public key and master key for each user can be determined. The attributes, public key and master key are denoted as

Attributes- $U = \{1, 2. . . N\}$

Public key- $PK = (Y, T_1, T_2, . . . , T_N)$

Master key- $MK = (y, t_1, t_2, . . . , t_N)$

where $T_i \in G_1$ and $t_i \in Z_p$ are for attribute i, $1 \leq i \leq N$, and $Y \in G_2$ is another public key component. We have $T_i = g^{ti}$ and $Y = e\,(g, g)^{y}$, $y \in Z_p$. While PK is publicly known to all the parties in the system, MK is kept as a secret by the authority party.

2) *Encryption:*

This is a randomized algorithm that takes a message M, the public key PK, and a set of attributes I as input. It outputs the cipher text E with the following format:

$$\mathbf{E = (I, \tilde{E}, \{E_{i \in I}\})}$$

where $\tilde{E} = MY^s$, $E_i = T_i^s$. and s is randomly chosen from Z_p

3) *Secret key generation:*

This is a randomized algorithm that takes as input an access tree T, the master key MK, and the public key K. It outputs a user secret key SK as follows. First, it defines a random polynomial $p_i(x)$ for each node i of T in the top-down manner starting from the root node r. For each non-root node j, $p_j(0) = p_{parent(j)}(idx(j))$ where $parent(j)$ represents j's parent and $idx(j)$ is j's unique index given by its parent. For the root node r, $p_r(0) = y$. Then it outputs SK as follows.

$$\mathbf{SK = \{sk_i\}_{i \in L}}$$

where L denotes the set of attributes attached to the leaf nodes of T and $sk_i = g^{pi(0)/ti}$.

4) *Decryption:*

IJERT

This algorithm takes as input the cipher text ***E*** encrypted under the attribute set **I**, the user's secret key ***SK*** for access tree ***T***, and the public key ***PK***. It first computes $e(E_i, sk_i) = e(g, g)^{pi(0)s}$ for leaf nodes. Then, it aggregates these pairing results in the bottom-up manner using the polynomial interpolation technique. Finally, it may recover the blind factor $Y_s=e(g,g)^{ys}$ and output the message ***M*** if and only if **I satisfies T**.

*5) **Access tree T:***

Let T be a tree representing an access structure. Each non-leaf node of the tree represents a threshold gate, described by its children and a threshold value. If num_x
is the number of children of a node x and k_x is its threshold value, then $0 < k_x \leq num_x$.
When $k_x = 1$, the threshold gate is an OR gate and when $k_x = num_x$, it is an AND gate.
Each leaf node x of the tree is described by an attribute and a threshold value $k_x = 1$.

To facilitate working with the access trees, we define a few functions. We denote the parent of the node x in the tree by parent(x). The function att(x) is defined only if x is a leaf node and denotes the attribute associated with the leaf node x in the tree. The access tree T also defines an ordering between the children of every node, that is, the children of a node are numbered from 1 to *num*. The function index(x) returns such a number associated with the node x, where the index values are uniquely assigned to nodes in the access structure for a given key in an arbitrary manner.

*6) **Satisfying an access tree:***

Let T be an access tree with root r. Denote by T_x the sub tree of T rooted at the node x. Hence T is the same as T_r. If a set of attributes I satisfies the access tree T_x, we denote it as $T_x(I) = 1$. We compute $T_x(I)$ recursively as follows. If x is a non-leaf node, evaluate $T_{x'}(I)$ for all children x' of node x. $T_x(I)$ returns 1 if and only if at least k_x children return 1. If x is a leaf node, then $T_x(I)$ returns 1 if and only if att(x) $\in I$.

*7) **Construction of Access Trees:***

In the access-tree construction, cipher texts are labeled with a set of descriptive attributes. Private keys are identified by a tree-access structure in which each interior node of the tree is a threshold gate and the leaves are associated with attributes. A user will be able to decrypt a cipher text with a given key if and only if there is an assignment of attributes from the cipher texts to nodes of the tree such that the tree is satisfied.

*B. **Proxy Re-Encryption (PRE)***

Proxy Re-Encryption (PRE) is a cryptographic primitive in which a semi-trusted proxy is able to convert a cipher text encrypted under Alice's public key into another cipher text that can be opened by Bob's private key without seeing the underlying plaintext. A Proxy Re-Encryption scheme allows the proxy, given the proxy re-encryption key $\boldsymbol{rk_{a\leftrightarrow b}}$, to translate cipher texts under public key $\boldsymbol{pk_a}$ into cipher texts under public key $\boldsymbol{pk_b}$ and vice versa [16].

First, we consider *protocol divertibility*, in which the (honest) intermediary, called a *warden*, randomizes all messages so that the intended underlying protocol succeeds, but information contained in subtle deviations from the protocol (for example, information coded into the values of supposedly random challenges) will be obliterated by the warden's transformation. Next, we introduce *atomic proxy cryptography*, in which two parties publish a *proxy key* that allows an untrusted intermediary to convert cipher texts encrypted for the first party directly into cipher texts that can be decrypted by the second. The intermediary learns neither clear text nor secret keys.

*1) **Divertible Protocols:***

The basic observation was that some 2-party identification protocols could be extended by placing an intermediary called a warden for historical reasons between the prover and verifier so that, even if both parties conspire, they cannot distinguish talking to each other through the warden from talking directly to a hypothetical honest verifier and honest prover, respectively.

In order to deal with protocols of more than two parties, we generalize the notion of *Interactive Turing machine* (ITM). Then we define connections of ITMs and finally give the definition of protocol divertibility.

*2) **(m, n)-Interactive Turing Machine:***

An (m, n)-*Interactive Turing Machine* ((m, n)-*ITM*) is a Turing machine with
$m \in \mathbb{N}$ read-only *input tapes*, m write-only *output tapes*, m read-only *random tapes*, a *work tape*, a read-only *auxiliary tape*, and $n \in \mathbb{N}_0$ pairs of *communication tapes*. Each pair consists of one read-only and one write-only tape that serves for reading in-messages from or writing out-messages to another ITM. (The purpose of allowing n=0 will become clear below.) The random tapes each contain an infinite stream of bits chosen uniformly at random.

IJERT

Read-only tapes are readable only from left to right. If the string to the right of a read-only head is empty, then we say the tape is *empty*.

Associated to an ITM is a *security parameter* $k \in \mathbb{N}$, a family $D = \{D_\pi\}_\pi$ of tuples of domains, a probabilistic *picking algorithm pick*(k) and an encoding scheme S. Each member

$$D_\pi = (In^{(1)}_\pi , \ldots , In^{(m)}_\pi , Out^{(1)}_\pi , \ldots , Out^{(m)}_\pi , \Omega^{(1)}_\pi , \ldots ,\Omega^{(m)}_\pi , (IM^{(1)}_\pi ,OM^{(1)}_\pi), \ldots , (IM^{(n)}_\pi ,OM^{(n)}_\pi))$$

of D contains one input (output, choice, in-message, out-message) domain for each of the m input (output, random) tapes and n (read-only, write-only) communication tapes. The algorithm *pick*(k) on input some security parameter k outputs a family index π. Finally, there is a polynomial $P(k)$ so that for each π chosen by *pick*(k), S encodes all elements of all domains in D_π as bitstrings of length $P(k)$.

3) *Atomic Proxy Cryptography:*

A basic goal of public-key encryption is to allow only the key or keys selected
at the time of encryption to decrypt the cipher text. To change the cipher text to a different key requires re-encryption of the message with the new key, which implies access to the original clear text and to a reliable copy of the new encryption key.

Here, on the other hand, we investigate the possibility of *atomic proxy functions* that convert ciphertext for one key into ciphertext for another without revealing secret decryption keys or cleartext messages .An atomic proxy function allows an untrusted party to convert ciphertext between keys without access to either the original message or to the secret component of the old key or the new key.

4) *Categories of Proxy Scheme:*

Symmetric proxy functions also imply that B trusts A, e.g., because d_B can be feasibly calculated given the proxy key plus d_A. *Asymmetric* proxy functions do not imply this bilateral trust.

In an *active asymmetric* scheme, B has to cooperate to produce the proxy key $\pi_{A\rightarrow B}$ feasibly, although the proxy key (even together with A's secret key) might not compromise B's secret key. In a *passive asymmetric* scheme, on the other hand, A's secret key and B's public key suffice to construct the proxy key.

Transparent proxy keys reveal the original two public keys to a third party. *Translucent* proxy keys allow a third party to verify a guess as to which two keys are involved (given their public keys). *Opaque* proxy keys reveal nothing, even to an adversary who correctly guesses the original public keys (but who does not know the secret keys involved).

C. *Lazy Re-Encryption (LRE)*

The lazy re-encryption technique and allow Cloud Servers to aggregate computation tasks of multiple operations. The operations such as

- Update secret keys
- Update user attributes.

Lazy re-encryption operates by using correlations in data updates to decide when to rekey. Since data re-encryption accounts for the larger part of the cost of key replacement, re-encryption is only performed if the data changes significantly after a user departs or if the data is highly sensitive and requires immediate re-encryption to prevent the user from accessing it. The cost of rekeying is minimized, but the problem remains of having to re-encrypt the data after a user's departure. Moreover, if a sensitive file does not change frequently, lazy re-encryption can allow a malicious user time to copy off information from the file into another file and leave the system without ever being detected. We have to assume that if a key k requires updating, then any objects encrypted with k are available to any user who could derive k. Hence, we may as well wait until the contents of an object changes before re-encrypting it. Similarly, we may as well defer sending a user u the new key k′ until such time as u actually requires k′ to decrypt an object. This is sometimes referred to as lazy update and lazy re-encryption.

A revoked reader who has access to the server will still have read access to the files not changed since the user's revocation, but will never be able to read data updated since their revocation. Lazy revocation, however, is complicated when multiple files are encrypted with the same key, as is the case when using filegroups. In this case, whenever a file gets updated, it gets encrypted with a new key. This causes filegroups to get fragmented (meaning a filegroup could have more than one key), which is undesirable. The next section describes how we mitigate this problem; briefly, we show how readers and writers can generate all the previous keys of a fragmented filegroup from the current key.

IJERT

V. CONCLUSION

Our paper aims at secured access control in cloud computing., which is not provided by current work. In this paper we propose a scheme to achieve this goal by exploiting all data backup and encryption techniques like KP-ABE and uniquely combining it with cryptographic techniques of proxy re-encryption and lazy re-encryption. Moreover, our proposed scheme can enable the data owner to delegate most of computation overhead to powerful cloud servers. Confidentiality of user access privilege and user secret key accountability can be achieved. Formal security proofs show that our proposed scheme is secure under standard cryptographic models and data storage discrepancies are removed .

VI. REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," University of California, Berkeley, Tech. Rep. USB-EECS-2009-28, Feb 2009.

[2] Amazon Web Services (AWS), online at http://aws.amazon.com.

[3] Google App Engine, Online at http://code.google.com/appengine/.

[4] Microsoft Azure, http://www.microsoft.com/azure/.

[5] 104th United States Congress, "Health Insurance Portability and Accountability Act of 1996 (HIPPA)," Online at http://aspe.hhs.gov/admnsimp/pl104191.htm, 1996.

[6] H. Harney, A. Colgrove, and P. D. McDaniel, "Principles of policy in secure groups," in *Proc. of NDSS'01*, 2001.

[7] P. D. McDaniel and A. Prakash, "Methods and limitations of security policy reconciliation," in *Proc. of SP'02*, 2002.

[8] T. Yu and M. Winslett, "A unified scheme for resource protection in automated trust negotiation," in *Proc. of SP'03*, 2003.

[9] J. Li, N. Li, and W. H. Winsborough, "Automated trust negotiation using cryptographic credentials," in *Proc. of CCS'05*, 2005.

[10] J. Anderson, "Computer Security Technology Planning Study," Air Force Electronic Systems Division, Report ESD-TR-73-51, 1972, http://seclab.cs.ucdavis.edu/projects/history/.

[11] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Scalable secure file sharing on untrusted storage," in *Proc. of FAST'03*, 2003.

[12] E. Goh, H. Shacham, N. Modadugu, and D. Boneh, "Sirius: Securing remote untrusted storage," in *Proc. of NDSS'03*, 2003.

[13] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," in *Proc. of NDSS'05*, 2005.

[14] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Over-encryption: Management of access control evolution on outsourced data," in *Proc. of VLDB'07*, 2007.

[15] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. Of CCS'06*, 2006.

[16] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *Proc. of EUROCRYPT '98*, 1998.

[17] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *Proc. of ESORICS '09*, 2009.

[18] L. Youseff, M. Butrico, and D. D. Silva, "Toward a unified ontology of cloud computing," in *Proc. of GCE'08*, 2008.

[19] S. Yu, K. Ren, W. Lou, and J. Li, "Defending against key abuse attacks in kp-abe enabled broadcast systems," in *Proc. of SECURECOMM'09*, 2009.

[20] D. Sheridan, "The optimality of a fast CNF conversion and its use with SAT," in *Proc. of SAT'04*, 2004.

[21] D. Naor, M. Naor, and J. B. Lotspiech, "Revocation and tracing schemes for stateless receivers," in *Proc. of CRYPTO'01*, 2001.

[22] M. Atallah, K. Frikken, and M. Blanton, "Dynamic and efficient key management for access hierarchies," in *Proc. of CCS'05*, 2005.

[23] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou, "Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing," in *Proc. of INFOCOM'10*, 2010.

IJERT