This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2010 proceedings
This paper was presented as part of the main Technical Program at IEEE INFOCOM 2010.

Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing

Shucheng Yu*, Cong Wang†, Kui Ren†, and Wenjing Lou*
*Dept. of ECE, Worcester Polytechnic Institute, Email: {yscheng, wjlou}@ece.wpi.edu
†Dept. of ECE, Illinois Institute of Technology, Email: {cong, kren}@ece.iit.edu

***Abstract*—Cloud computing is an emerging computing paradigm in which resources of the computing infrastructure are provided as services over the Internet. As promising as it is, this paradigm also brings forth many new challenges for data security and access control when users outsource sensitive data for sharing on cloud servers, which are not within the same trusted domain as data owners. To keep sensitive user data confidential against untrusted servers, existing solutions usually apply cryptographic methods by disclosing data decryption keys only to authorized users. However, in doing so, 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. The problem of simultaneously achieving fine-grainedness, scalability, and data confidentiality of access control actually still remains unresolved. This paper addresses this challenging open issue by, on one hand, defining and enforcing access policies based on data attributes, and, on the other hand, allowing the data owner to delegate most of the computation tasks involved in fine-grained data access control to untrusted cloud servers without disclosing the underlying data contents. We achieve this goal by exploiting and uniquely combining techniques of attribute-based encryption (ABE), proxy re-encryption, and lazy re-encryption. Our proposed scheme also has salient properties of user access privilege confidentiality and user secret key accountability. Extensive analysis shows that our proposed scheme is highly efficient and provably secure under existing security models.**

I. Introduction

Cloud computing is a promising computing paradigm which recently has drawn extensive attention from both academia and industry. By combining a set of existing and new 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. Along with this new paradigm, various business models are developed, which can be described by terminology of "X as a service (XaaS)" [1] where X could be software, hardware, data storage, and etc. Successful examples are Amazon's EC2 and S3 [2], Google App Engine [3], and Microsoft Azure [4] which provide users with scalable resources in the pay-as-you-use fashion at relatively low prices. For example, Amazon's S3 data storage service just charges \$0.12 to \$0.15 per gigabyte-month. 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 promising as it is, cloud computing is also facing many challenges that, if not well resolved, may impede its fast growth. Data security, as it exists in many other applications, is among these challenges that would raise great concerns from users when they store sensitive information on cloud servers. These concerns originate from the fact that cloud servers are usually operated by commercial providers which are very likely to be outside of the trusted domain of the users. Data confidential against cloud servers is hence frequently desired when users outsource data for storage in the cloud. In some practical application systems, data confidentiality is not only a security/privacy issue, but also of juristic concerns. For example, in healthcare application scenarios use and disclosure of protected health information (PHI) should meet the requirements of Health Insurance Portability and Accountability Act (HIPAA) [5], and keeping user data confidential against the storage servers is not just an option, but a requirement.

Furthermore, we observe that there are also cases in which cloud users themselves are content providers. They publish data on cloud servers for sharing and need fine-grained data access control in terms of which user (data consumer) has the access privilege to which types of data. In the healthcare case, for example, a medical center would be the data owner who stores millions of healthcare records in the cloud. It would allow data consumers such as doctors, patients, researchers and etc, to access various types of healthcare records under policies admitted by HIPAA. To enforce these access policies, the data owners on one hand would like to take advantage of the abundant resources that the cloud provides for efficiency and economy; on the other hand, they may want to keep the data contents confidential against cloud servers.

As a significant research area for system protection, data access control has been evolving in the past thirty years and various techniques [6]–[9] have been developed to effectively implement fine-grained access control, which allows flexibility in specifying differential access rights of individual users. Traditional access control architectures usually assume the data owner and the servers storing the data are in the same trusted domain, where the servers are fully entrusted as an omniscient reference monitor [10] responsible for defining and enforcing access control policies. This assumption however no longer holds in cloud computing since the data owner and cloud servers are very likely to be in two different domains. On one hand, cloud servers are not entitled to access the outsourced data content for data confidentiality; on the other hand, the data resources are not physically under the full control of

978-1-4244-5837-0/10/$26.00 ©2010 IEEE

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2010 proceedings
This paper was presented as part of the main Technical Program at IEEE INFOCOM 2010.

the owner. For the purpose of helping the data owner enjoy fine-grained access control of data stored on untrusted cloud servers, a feasible solution would be encrypting data through certain cryptographic primitive(s), and disclosing decryption keys only to authorized users. Unauthorized users, including cloud servers, are not able to decrypt since they do not have the data decryption keys. This general method actually has been widely adopted by existing works [11]–[14] which aim at securing data storage on untrusted servers. One critical issue with this branch of approaches is how to achieve the desired security goals without introducing a high complexity on key management and data encryption. These existing works, as we will discuss in section V-C, resolve this issue either by introducing a per file access control list (ACL) for fine-grained access control, or by categorizing files into several $filegroups$ for efficiency. As the system scales, however, the complexity of the ACL-based scheme would be proportional to the number of users in the system. The $filegroup$-based scheme, on the other hand, is just able to provide coarse-grained data access control. It actually still remains open to simultaneously achieve the goals of fine-grainedness, scalability, and data confidentiality for data access control in cloud computing.

In this paper, we address this open issue and propose a secure and scalable fine-grained data access control scheme for cloud computing. Our proposed scheme is partially based on our observation that, in practical application scenarios each data file can be associated with a set of attributes which are meaningful in the context of interest. The access structure of each user can thus be defined as a unique logical expression over these attributes to reflect the scope of data files that the user is allowed to access. 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 ciphertext if and only if the data file attributes satisfy his access structure. Such a design also brings about the efficiency benefit, as compared to previous works, in that, 1) the complexity of encryption is just related the number of attributes associated to the data file, and is independent to the number of users in the system; and 2) data file creation/deletion and new user grant operations just affect current file/user without involving system-wide data file update or re-keying. One 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 (KP-ABE) [15], and uniquely combine it with the technique of proxy re-encryption (PRE) [16] and lazy re-encryption [11].

Main contributions of this paper can be summarized as follows. 1) To the best of our knowledge, this paper is the first that simultaneously achieves fine-grainedness, scalability and data confidentiality for data access control in cloud computing; 2) Our proposed scheme enables the data owner to delegate most of computation intensive tasks to cloud servers without disclosing data contents or user access privilege information; 3) The proposed scheme is provably secure under the standard security model. In addition, our proposed scheme is able to support user accountability with minor extension.

The rest of this paper is organized as follows. Section II discusses models and assumptions. Section III reviews some technique preliminaries pertaining to our construction. Section IV presents our construction. In section V, we analyze our proposed scheme in terms of its security and performance. We conclude this paper in Section VI.

II. Models and Assumptions

A. System Models

Similar to [17], 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 [12] 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 recently proposed by Youseff et al. [18].

B. Security Models

In this work, we just consider Honest but Curious Cloud Servers as [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,

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2010 proceedings
This paper was presented as part of the main Technical Program at IEEE INFOCOM 2010.

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. Technique Preliminaries

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 ciphertext if and only if the data attributes satisfy his access structure. A KP-ABE scheme is composed of four algorithms which can be defined as follows:

Setup This algorithm takes as input a security parameter κ and the attribute universe $U = \{1, 2, \ldots, N\}$ of cardinality N. It defines a bilinear group $\mathbb{G}_1$ of prime order p with a generator g, a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ which has the properties of *bilinearity*, *computability*, and *non-degeneracy*. It returns the public key PK as well as a system master key MK as follows

$$PK = (Y, T_1, T_2, \ldots, T_N)$$
$$MK = (y, t_1, t_2, \ldots, t_N)$$

where $T_i \in \mathbb{G}_1$ and $t_i \in \mathbb{Z}_p$ are for attribute i, $1 \leq i \leq N$, and $Y \in \mathbb{G}_2$ is another public key component. We have $T_i = g^{t_i}$ and $Y = e(g, g)^y$, $y \in \mathbb{Z}_p$. While PK is publicly known to all the parties in the system, MK is kept as a secret by the authority party.

Encryption This algorithm takes a message M, the public key PK, and a set of attributes I as input. It outputs the ciphertext E with the following format:

$$E = (I, \tilde{E}, \{E_i\}_{i \in I})$$

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

Key Generation This algorithm takes as input an access tree T, the master key MK, and the public key PK. 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.

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

where L denotes the set of attributes attached to the leaf nodes of T and $sk_i = g^{\frac{p_i(0)}{t_i}}$.

Decryption This algorithm takes as input the ciphertext 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 $e(E_i, sk_i) = e(g, g)^{p_i(0)s}$ 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.

Please refer to [15] for more details on KP-ABE algorithms. [19] is an enhanced KP-ABE scheme which supports user secret key accountability.

B. Proxy Re-Encryption (PRE)

Proxy Re-Encryption (PRE) is a cryptographic primitive in which a semi-trusted proxy is able to convert a ciphertext encrypted under Alice's public key into another ciphertext that can be opened by Bob's private key without seeing the underlying plaintext. More formally, a PRE scheme allows the proxy, given the proxy re-encryption key $rk_{a \leftrightarrow b}$, to translate ciphertexts under public key pk_a into ciphertexts under public key pk_b and vise versa. Please refer to [16] for more details on proxy re-encryption schemes.

IV. Our Proposed Scheme

A. Main Idea

In order to achieve secure, scalable and fine-grained access control on outsourced data in the cloud, we utilize and uniquely combine the following three advanced cryptograhphic techniques: KP-ABE, PRE and lazy re-encryption. 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 inevitabily 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

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2010 proceedings
This paper was presented as part of the main Technical Program at IEEE INFOCOM 2010.

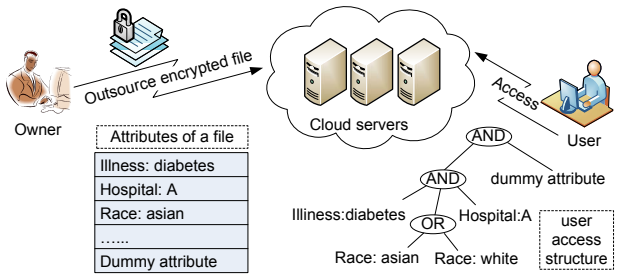

Fig. 1: An examplary case in the healthcare scenario

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. Accoutability of user secret key can also be achieved by using an enhanced scheme of KP-ABE.

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 IDs 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	ciphertext 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. 2: 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 $ASetup(\kappa)$, 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 $DEK \xleftarrow{R} \mathcal{K}$, where $\mathcal{K}$ 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., $(\tilde{E}, \{E_i\}_{i\in I}) \leftarrow AEncrypt(I, DEK, PK)$.

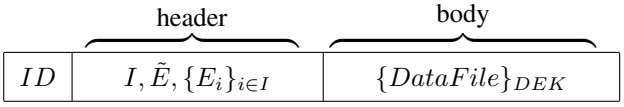

Fig. 3: Format of a data file stored on the cloud

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.

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2010 proceedings
This paper was presented as part of the main Technical Program at IEEE INFOCOM 2010.

// to revoke user v

// stage 1: attribute update.

The Data Owner		**Cloud Servers**
1. $D \leftarrow AMinimalSet(P)$, where P is v's access structure;		remove v from the system user list UL;
2. for each attribute i in D		for each attribute $i \in D$
$(t'_i, T'_i, rk_{i \hookrightarrow i'}) \leftarrow$ *AUpdateAtt*(i, MK);	$\xrightarrow{Att}$	store $(i, T'_i, \delta_{O,(i,T'_i)})$;
3. send $Att = (v, D, \{i, T'_i, \delta_{O,(i,T'_i)}, rk_{i \hookrightarrow i'}\}_{i \in D})$.		add $rk_{i \hookrightarrow i'}$ to i's history list AHL_i.

// stage 2: data file and user secret key update.

Cloud Servers		**User**(u)
1. on receiving REQ, proceed if $u \in UL$;		
2. get the tuple $(u, \{j, sk_j\}_{j \in L_P \backslash Att_D})$;		1. generate data file access request REQ;
for each attribute $j \in L_P \backslash Att_D$	$\xleftarrow{REQ}$	2. wait for the response from Cloud Servers;
$sk'_j \leftarrow$ *AUpdateSK*(j, sk_j, AHL_j);		
for each requested file f in REQ		3. on receiving $RESP$, verify each $\delta_{O,(j,T'_j)}$
for each attribute $k \in I_f$	$\xrightarrow{RESP}$	and sk'_j; proceed if all correct;
$E'_k \leftarrow$ A*UpdateAtt4File*(k, E_k, AHL_k);		4. replace each sk_j in SK with sk'_j;
3. send $RESP = (\{j, sk'_j, T'_j, \delta_{O,(j,T'_j)}\}_{j \in L_P \backslash Att_D}, FL)$.		5. decrypt each file in FL with SK.

Fig. 4: Description of the process of user revocation

- 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 ciphertext by C;
- send the tuple $(T, C, \delta_{O,(T,C)})$ to Cloud Servers, where T denotes the tuple $(w, \{j, sk_j\}_{j \in L_P \backslash Att_D})$.

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 can not 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, DEKs 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 in Fig.4.

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

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2010 proceedings
This paper was presented as part of the main Technical Program at IEEE INFOCOM 2010.

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 DEKs 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 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. Fig.5 depicts two of the four algorithms.

$AUpdateAtt(i, MK)$
randomly pick $t'_i \xleftarrow{R} \mathbb{Z}_p$;
compute $T'_i \leftarrow g^{t'_i}$, and $rk_{i \leftrightarrow i'} \leftarrow \frac{t'_i}{t_i}$;
output t'_i, T'_i, and $rk_{i \leftrightarrow i'}$.

$AUpdateAtt4File(i, E_i, AHL_i)$
if i has the latest version, exit;
search AHL_i and locate the old version of i;
// assume the latest definition of i in MK is $t_{i^{(n)}}$.
$rk_{i \leftrightarrow i^{(n)}} \leftarrow rk_{i \leftrightarrow i'} \cdot rk_{i' \leftrightarrow i''} \cdots rk_{i^{(n-1)} \leftrightarrow i^{(n)}} = \frac{t_{i^{(n)}}}{t_i}$;
compute $E_i^{(n)} \leftarrow (E_i)^{rk_{i \leftrightarrow i^{(n)}}} = g^{t_{i^{(n)}} s}$;
output $E_i^{(n)}$.

Fig. 5: Pseudo-code of algorithm level 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 $E_i^{(n)}$ 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.

D. Summary

In our proposed scheme, we exploit the technique of hybrid encryption to protect data files, i.e., we encrypt data files using symmetric DEKs and encrypt DEKs with KP-ABE. Using KP-ABE, we are able to immediately enjoy fine-grained data access control and efficient operations such as file creation/deletion and new user grant. To resolve the challenging issue of user revocation, we combine the technique of proxy re-encryption with KP-ABE and delegate most of the burdensome computational task to Cloud Servers. We achieve this by letting Cloud Servers keep a partial copy of each user's secret key, i.e., secret key components of all but one (dummy) attributes. When the data owner redefines a certain set of attributes for the purpose of user revocation, he also generates corresponding proxy re-encryption keys and sends them to Cloud Servers. Cloud Servers, given these proxy re-encryption keys, can update user secret key components and re-encrypt data files accordingly without knowing the underlying plaintexts of data files. This enhancement releases the data owner from the possible huge computation overhead on user revocation. The data owner also does not need to always stay online since Cloud Servers will take over the burdensome task after having obtained the PRE keys. To further save computation overhead of Cloud Servers on user revocation, we use the technique of lazy re-encryption and enable Cloud Servers to "aggregate" multiple successive secret key update/file re-encryption operations into one, and thus statistically save the computation overhead.

V. Analysis of Our Proposed Scheme

A. Security Analysis

We first analyze security properties of our proposed scheme, starting with the following immediately available properties.

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2010 proceedings
This paper was presented as part of the main Technical Program at IEEE INFOCOM 2010.

1) Fine-grainedness of Access Control: In our proposed scheme, the data owner is able to define and enforce expressive and flexible access structure for each user. Specifically, the access structure of each user is defined as a logic formula over data file attributes, and is able to represent any desired data file set.

2) User Access Privilege Confidentiality: Our proposed scheme just discloses the leaf node information of a user access tree to Cloud Servers. As interior nodes of an access tree can be any threshold gates and are unknown to Cloud Servers, it is hard for Cloud Servers to recover the access structure and thus derive user access privilege information.

3) User Secret Key Accountability: This property can be immediately achieved by using the enhanced construction of KP-ABE [19] which can be used to disclose the identities of key abusers.

Now we analyze data confidentiality of our proposed scheme by giving a cryptographic security proof.

4) Data Confidentiality: We analyze data confidentiality of our proposed scheme by comparing it with an intuitive scheme in which data files are encrypted using symmetric DEKs, and DEKs are direclty encrypted using standard KP-ABE. In this intuitive scheme just ciphertexts of data files are given to Cloud Servers. Assuming the symmetric key algorithm is secure, e.g., using standard symmtric key algorithm such as AES, security of this intuitive scheme is merely relied on the security of KP-ABE. Actually, the standard KP-ABE is provably secure under the attribute-based Selective-Set model [15] given the Decisional Bilinear Diffie-Hellman (DBDH) problem is hard. Therefore, the intuitive scheme is secure under the same model. Our goal is to show that our proposed scheme is as secure as the intuitive scheme. As compared to the intuitive scheme, our scheme discloses the following extra information to Cloud Servers: a partial set of user secret key components (except for the one for the dummy attribute which is required for each decryption), and the proxy re-encryption keys. Based on this observation, we sketch the security proof of our proposed scheme using a series of games as follows.

Game 0: This is the security game of the intuitive scheme.

Game 1: The difference between this game and *Game 0* is that, in this game more than one (MK, PK) pairs are defined, and the adversary is given the PK's as well as the secret keys for each access structure he submits. In addition, the adversary is also given the proxy re-encryption keys (between any two (MK, PK) pairs).

Game 2: This game is for our proposed scheme. The only difference between this game and *Game 1* is that, in this game the partial set of user secret key components are disclosed to the adversary.

Lemma 5.1: The advantage of the adversary in *Game 1* is the same as that in *Game 0*.

Proof: Our first goal is to show that, if there is a polynomial time algorithm $\mathcal{A}$ that wins the semantic security game of *Game 1* with non-negligible advantage, we can use it to build a polynomial time algorithm $\mathcal{B}$ to win the semantic security game of *Game 0*, i.e., the game under the attribute-based Selective-Set model. In the semantic security game, the adversary submits two equal length challenge message m_0 and m_1. The challenger flips a random coin $b \leftarrow \{0,1\}$ and encrypts m_b. The challenge cihpertext E is then given to the adversary. The adversary is asked to output his guess b' of the random coin b. If $b' = b$ the adversary wins. During this game, the adversary is given public parameters and allowed to query many user secret keys except for the one for the challenge ciphertext. Assuming the algorithm $\mathcal{A}$ (i.e., the adversary) can win the semantic security game, i.e., $b \leftarrow \mathcal{A}(E_A, \{PK_i\}_{1\le i\le k}, \{SK_j\}_{1\le j\le q_S}, \{rk\})$, with non-negligible advantage, where $\{PK_i\}$ and $\{SK_j\}$ denotes the set of all PK's and the set of all the secret keys given to $\mathcal{A}$ respectively, $\{rk\}$ representing the set of all the proxy re-encryption keys, q_S denoting the number of secret key queries, and k representing the number of PK's, A polynomial time algorithm $\mathcal{B}$ can be built as is shown in Fig.6. Therefore, the advantage of the adversary in *Game 1* is not higher than that in *Game 0*. However, the advantage of the adversary in *Game 1* can not be lower than that in *Game 0* since the adversary is given more information in *Game 1* than in *Game 0*. Therefore, the advantages in the two games are the same.

$\mathcal{B}(E_B, PK, \{SK'_j\}_{1\le j\le q_S})$
assume $PK = (Y, T_1, T_2, \cdots, T_N)$;
$E_B = (I, \tilde{E}, \{E_j\}_{j\in I})$;
$SK'_j = \{sk_{j,i}\}_{i\in L}$ for all $j \in \{1, \cdots, q_S\}$;
for u from 1 to k:
for v from 1 to N:
random choose $r_{uv} \overset{R}{\leftarrow} \mathbb{Z}_p$;
if $u > 1$, $rk_{u-1\leftrightarrow u} = \frac{r_{(u-1)v}}{r_{uv}}$;
add $rk_{u-1\leftrightarrow u}$ to {rk};
$PK_u = (Y, T_1^{r_{u1}}, T_2^{r_{u2}}, \cdots, T_N^{r_{uN}})$;
$E_A \leftarrow (I, \tilde{E}, \{E_j^{r_{uj}}\}_{j\in I})$, where $u \overset{R}{\leftarrow} \{1, \cdots, k\}$;
$SK_j = \{(sk_{j,i})^{\frac{1}{r_{ui}}}\}_{i\in L}$, u is the same as above;
$b' \leftarrow \mathcal{A}(E_A, \{PK_i\}_{1\le i\le k}, \{SK_j\}_{1\le j\le q_S}, \{rk\})$.

Fig. 6: Construction of Algorithm $\mathcal{B}$ from $\mathcal{A}$

Notably, the chosen ciphertext security of our proposed scheme can also be proved similarly since any decryption oracle submitted by $\mathcal{A}$ can be forwarded by $\mathcal{B}$ to the challenger of *Game 0*, and the answers can be then forwarded to $\mathcal{A}$. ■

Lemma 5.2: The advantage of the adversary in *Game 2* is the same as that in *Game 1*.

Proof: As described previously, the extra information disclosed to the adversary in *Game 2* are the partial user secret keys. These partial user secret keys are actually equivalent to the secret keys queried by the adversary in *Game 1*. Therefore, the view of the adversary in the two games are the same. This proves this lemma. ■

According to the above two lemmas, we can conclude that our proposed scheme is as secure as the intuitive scheme, which is provably secure. This proves data confidentiality of our proposed scheme, even under collusion attacks between Cloud Servers and malicious users. This is because in our security game, the adversary $\mathcal{A}$ has the same capability as Cloud Servers who are given many secret keys of unauthorized

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2010 proceedings
This paper was presented as part of the main Technical Program at IEEE INFOCOM 2010.

users.

B. Performance Analysis

This section numerically evaluates the performance of our proposed scheme in terms of the computation overhead introduced by each operation as well as the ciphertext size.

1) Computation Complexity: We analyze the computation complexity for the following six operations: *system setup*, *new file creation*, *file deletion*, *new user grant*, *user revocation*, and *file access*.

System Setup In this operation, the data owner needs to define underlying bilinear groups, and generate PK and MK. As is described in Section III-A, the main computation overhead for the generation of PK and MK is introduced by the N group multiplication operations on $\mathbb{G}_1$.

New File Creation The main computation overhead of this operation is the encryption of the data file using the symmetric DEK as well as the encryption of the DEK using KP-ABE. The complexity of the former depends on the size of the underlying data file and inevitable for any cryptographic method. The computation overhead for the latter consists of $|I|$ multiplication operations on $\mathbb{G}_1$ and 1 multiplication operation on $\mathbb{G}_2$, where I denotes the attribute set I of the data file. All these operations are for the data owner.

File Deletion This operation just involves the data owner and Cloud Servers. The former needs to compute one signature and the latter verifies this signature.

New User Grant This operation is executed interactively by the data owner, Cloud Servers, and the user. The computation overhead for the data owner is mainly composed of the generation of the user secret key and encryption of the user secret key using the user's public key. The former accounts for $|L|$ multiplication operations on $\mathbb{G}_1$, where L denotes the set of leaf nodes of the access tree. The latter accounts for one PKC operation, e.g., RSA encryption. The main overhead for Cloud Servers is one signature verification. The user needs to do two PKC operations, one for data decryption and the other for signature verification.

User Revocation This operation is composed of two stages. The second stage can actually be amortized as the file access operation. Here we just counts the operation overhead for the first stage. That for the second stage will be included in the file access operation. The first stage occurs between the data owner and Cloud Servers. The computation overhead for the data owner is caused by the execution of $AMinimalSet$ and $AUpdateAtt$ as well as the generation of his signatures for the public key components. The complexity of algorithm $AMinimalSet$ is actually mainly contributed by the CNF conversion operation which can be efficiently realized by existing algorithms such as [20] (with the complexity linear to the size of the access structure). Assuming the size of the minimal set returned by $AMinimalSet$ is D, $D \leq N$, the computation overhead for $AUpdateAtt$ is mainly contributed by D multiplication operations on $\mathbb{G}_1$. In addition, the data owner also needs to compute D signatures on public key components. The computation overhead on Cloud Servers in this stage is negligible. When counting the complexity of user revocation, we use N instead of the size of the access structure since in practical scenarios $AMinimalSet$ is very efficient if we limit the size of access structure (without affecting system scalability), but each signature or multiplication operation on $\mathbb{G}_1$ is expensive.

File Access This operation occurs between Cloud Servers and the user. For Cloud Servers, the main computation overhead is caused by the execution of algorithm $AUpdateSK$ and algorithm $AUpdateAtt4File$. In the worst case, the algorithm $AUpdateSK$ would be called $|L|-1$ times, which represents $|L|-1$ multiplication operations on $\mathbb{G}_1$. Each execution of the algorithm $AUpdateAtt4File$ accounts for one multiplication operation on $\mathbb{G}_1$. In the worst case, Cloud Servers need to call $AUpdateAtt4File$ N times per file access. Our lazy re-encryption solution will greatly reduce the average system-wide call times of these two algorithms from statistical point of view. File decryption needs $|L|$ bilinear pairing in the worst case. Fig.7 summarizes the computation complexity of our proposed scheme.

Operation	Complexity
File Creation	$\mathcal{O}(\|I\|)$
File Deletion	$\mathcal{O}(1)$
User Grant	$\mathcal{O}(\|L\|)$
User Revocation	$\mathcal{O}(N)$
File Access	$\mathcal{O}(max(\|L\|, N))$

Fig. 7: Complexity of our proposed scheme

2) Ciphertext Size: As is depicted in Section IV-C, the ciphertext is composed of an ID, a header, and a body. The body is just the data block. The header for each data file is composed of an attribute set I, one group element on $\mathbb{G}_2$, and $|I|$ group elements on $\mathbb{G}_1$.

C. Related Work

Existing work close to ours can be found in the areas of "shared cryptographic file systems" and "access control of outsourced data".

In [11], Kallahalla et al proposed Plutus as a cryptographic file system to secure file storage on untrusted servers. Plutus groups a set of files with similar sharing attributes as a file-group and associates each file-group with a symmetric lockbox-key. Each file is encrypted using a unique file-blcok key which is further encrypted with the lockbox-key of the file-group to which the file belongs. If the owner wants to share a file-group, he just delivers the corresponding lockbox-key to users. As the complexity of key management is proportional to the total number of file-groups, Plutus is not suitable for the case of fine-grained access control in which the number of possible "file-groups" could be huge.

In [12], Goh et al proposed SiRiUS which is layered over existing file systems such as NFS but provides end-to-end security. For the purpose of access control, SiRiUS attaches each file with a meta data file that contains the file's access control list (ACL), each entry of which is the encryption of

the file's file encryption key (FEK) using the public key of an authorized user. The extension version of SiRiUS uses NNL broadcast encryption algorithm [21] to encrypt the FEK of each file instead of encrypting it with each individual user's public key. As the complexity of the user revocation solution in NNL is proportional to the number of revoked users, SiRiUS has the same complexity in terms of each meta data file's size and the encryption overhead, and thus is not scalable.

Ateniese et al [13] proposed a secure distributed storage scheme based on proxy re-encryption. Specifically, the data owner encrypts blocks of content with symmetric content keys. The content keys are all encrypted with a master public key, which can only be decrypted by the master private key kept by the data owner. The data owner uses his master private key and user's public key to generate proxy re-encryption keys, with which the semi-trusted server can then convert the ciphertext into that for a specific granted user and fulfill the task of access control enforcement. The main issue with this scheme is that collusion between a malicious server and any single malicious user would expose decryption keys of all the encrypted data and compromise data security of the system completely. In addition, user access privilege is not protected from the proxy server. User secret key accountability is neither supported.

In [14], Vimercati et al proposed a solution for securing data storage on untrusted servers based on key derivation methods [22]. In this proposed scheme, each file is encrypted with a symmetric key and each user is assigned a secret key. To grant the access privilege for a user, the owner creates corresponding public tokens from which, together with his secret key, the user is able to derive decryption keys of desired files. The owner then transmits these public tokens to the semi-trusted server and delegates the task of token distribution to it. Just given these public tokens, the server is not able to derive the decryption key of any file. This solution introduces a minimal number of secret key per user and a minimal number of encryption key for each file. However, the complexity of operations of file creation and user grant/revocation is linear to the number of users, which makes the scheme unscalable. User access privilege accountability is also not supported.

D. Discussion

According to the above analysis, we can see that our proposed scheme is able to realize the desired security goals, i.e., fine-grained access control, data confidentiality, user access privilege confidentiality, and user secret key accountability. The goal of scalability is also achieved since the complexity for each operation of our proposed scheme, as is shown in Fig. 7, is no longer dependent to the nunber of users in the system. Therefore, our proposed scheme can serve as an ideal candidate for data access control in the emerging cloud computing environment. On the contrary, existing access control schemes in related areas either lack scalability [12], [14] and fine-grainedness [11], or do not provide adequate proof of data confidentiality [13].

VI. Conclusion

This paper aims at fine-grained data access control in cloud computing. One challenge in this context is to achieve fine-grainedness, data confidentiality, and scalability simultaneously, which is not provided by current work. In this paper we propose a scheme to achieve this goal by exploiting KP-ABE and uniquely combining it with 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.

Acknowledgments

This work was supported in part by the US National Science Foundation under grants CNS-0716306, CNS-0831628, CNS-0746977, and CNS-0831963.

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.