Privacy-Preserving Fine-Grained Access Control in Public Clouds

Mohamed Nabeel, Elisa Bertino
{nabeel, bertino}@cs.purdue.edu
Department of Computer Science and Cyber Center
Purdue University

Abstract

With many economical benefits of cloud computing, many organizations have been considering moving their information systems to the cloud. However, an important problem in public clouds is how to selectively share data based on fine-grained attribute based access control policies while at the same time assuring confidentiality of the data and preserving the privacy of users from the cloud. In this article, we briefly discuss the drawbacks of approaches based on well known cryptographic techniques in addressing such problem and then present two approaches that address these drawbacks with different trade-offs.

1 Introduction

With the advent of technologies such as cloud computing, sharing data through a third-party cloud service provider has never been more economical and easier than now. However, such cloud providers cannot be trusted to protect the confidentiality of the data. In fact, data privacy and security issues have been major concerns for many organizations utilizing such services. Data often contains sensitive information and should be protected as mandated by various organizational policies and legal regulations. Encryption is a commonly adopted approach to assure data confidentiality. Encryption alone however is not sufficient as organizations often have also to enforce fine-grained access control on the data. Such control is often based on security-relevant properties of users, referred to as *identity attributes*, such as the roles of users in the organization, projects on which users are working, and so forth. These access control systems are referred to as *attribute based access control (ABAC) systems*. Therefore, an important requirement is to support fine-grained access control, based on policies specified using identity attributes, over encrypted data.

With the involvement of the third-party cloud services, a crucial issue is that the identity attributes in the access control policies may reveal privacy-sensitive information about users and organizations and leak confidential information about the content. The confidentiality of the content and the privacy of the users are thus not assured if the identity attributes are not protected. It is well-known that privacy, both individual as well as organizational, is considered a key requirement in all solutions, including cloud services, for digital identity management [7]. Further, as insider threats are one of the major sources of data theft and privacy breaches, identity attributes must be strongly protected even from accesses within organizations. With initiatives such as cloud computing the scope of insider threats is no longer limited to the organizational perimeter. Therefore, protecting the identity attributes of the users while enforcing attribute-based access control both within the organization as well as in the cloud is crucial.

For example, let us consider a hospital that decides to use the cloud to manage their electronic health record (EHR) system. Since EHRs are sensitive information, their confidentiality should be preserved from the cloud. A typical hospital stakeholders consist of employees playing different roles such as receptionist, cashier, doctor, nurse, pharmacist, system administrator, and so on. A cashier, for example, does not need have access to data in EHRs except the billing information in them while a doctor or a nurse does not need have access to billing information in EHRs. This requires the cloud based EHR system to support fine-grained access control. The typical identity attributes used by the stakeholders in our EHR system, such as role, location and position, can be used as good contextual information to connect with other publicly available information in order to learn sensitive information about individuals, leading to privacy violations. For example, if system administrators of the EHR system can see hospital employees' identity attributes, they can misuse the system to access EHRs and sell to outsiders without being caught. In order to address these issues, the cloud based EHR system should protect the identity attributes of users.

The goal of this article is to provide an overview of our approaches to enforce fine-grained access control on sensitive data stored in untrusted public clouds, while at the same assuring the confidentiality of the data from the cloud and preserving the privacy of users who are authorized to access the data. We compare these approaches and discuss about open issues.

The article is organized as follows. Section 2 briefly discusses the drawbacks of existing cryptographic techniques and presents a new approach for managing group encryption keys. Based on such new key management approach, Sections 3 and 4 present a basic approach and a two layer encryption-based approach for privacy-preserving ABAC for data on clouds, respectively. Section 5 compares the existing and new approaches. Finally, Section 3 outlines a few conclusions.

2 A New Approach to Manage Group Encryption Keys

An approach to support fine-grained selective ABAC is to identify the sets of data items to which the same access control policy (or set of policies) applies and then encrypt each such set with the same encryption key. The encrypted data is then uploaded to the cloud and each user is given the keys only for the set(s) of data items that it can access according to the policies [1]. Such approach addresses two requirements: (a) protecting data confidentiality from the cloud; (b) enforcing fine-grained access control policies with respect to the data users. A major issue in such an approach is represented by key management, as each user must be given the correct keys with respect to the access control policies that the user satisfies. One approach to such issue is to use a hybrid solution whereby the data encryption keys are encrypted using a public key cryptosystem such as attribute based encryption (ABE) [1] and/or proxy re-encryption (PRE) [10] [11]. However, such an approach has several weaknesses: it cannot efficiently handle adding/revoking users or identity attributes, and policy changes; it requires to keep multiple encrypted copies of the same key; it incurs high computational costs; it requires additional attributes to support revocation [2]. Therefore, a different approach is required.

It is also worth noting that a simplistic group key management (GKM) scheme by which the content publisher directly delivers the symmetric keys to the corresponding users has some major drawbacks with respect to user privacy and key management. On one hand, user private information encoded in the user identity attributes is not protected in the simplistic approach. On the other hand, such a simplistic key management scheme does not scale well when the number of users becomes large and multiple keys need to be distributed to multiple users. The goal of our work is to develop an approach which does not have these shortcomings.

[1] Here and the rest of the article the term user may indicate a real human user or some client software running on behalf of some human user.

We observe that, without utilizing public key cryptography and by allowing users to dynamically derive the symmetric keys at the time of decryption, one can address the above issues. Based on this idea, we have defined a new GKM scheme, called broadcast GKM (BGKM), and given a secure construction of the BGKM scheme [6]. The idea is to give secrets to users based on the identity attributes they have and later allow them to derive actual symmetric keys based on their secrets and some public information. A key advantage of the BGKM scheme is that adding users/revoking users or updating access control policies can be performed efficiently and only requires updating the public information. Our BGKM scheme is referred to as access control vector BGKM (ACV-BGKM). The idea of ACV-BGKM is to construct a special matrix A where each row is linearly independent and generated using each user's secret. The group controller generates the null space Y of this matrix by solving the linear system $AY = 0$, randomly selects a vector in the null space, and hides the group symmetric key inside this vector. We call this vector as access control vector (ACV) and is part of the public information. An authorized user can generate a vector in the row space of the special matrix using its secret and some public information. We call this vector as key extraction vector (KEV). The system is designed such that the inner product of ACV and KEV allows authorized users to derive the group symmetric key. We show that a user who does not have a valid secret has a negligible probability of generating a valid KEV and deriving the group key. When a user is revoked, the group controller simply updates the special matrix excluding the revoked user and generate a new ACV hiding a new group key. Notice that such revocation handling does not affect the existing users as only the public information is changed.

Using the ACV-BGKM scheme as a building block, we have constructed a more expressive scheme called attribute based GKM (AB-GKM) [5]. The idea is to generate an ACV-BGKM instance for each attribute and combine the instances together using an access structure that represents the attribute based access control policy. The AB-GKM scheme satisfies all the properties of the ACV-BGKM scheme and consists of the following five algorithms: **Setup**, **SecGen**, **KeyGen**, **KeyDer**and **Update**.

- **Setup(ℓ)**: It initializes the BGKM scheme using a security parameter ℓ. It also initializes the set of used secrets **S**, the secret space $\mathcal{SS}$, and the key space $\mathcal{KS}$.

- **SecGen(user, attribute)**: It picks a random bit string $s \notin$ **S** uniformly at random from $\mathcal{SS}$, adds s to **S** and outputs s. A unique secret is assigned to per user per attribute. These secrets are used by the group controller to generate the group key and also by the users to derive the group key.

- **KeyGen(S, Policy)**: It picks a group key k uniformly at random from the key space $\mathcal{KS}$ and outputs the public information tuple PI computed from the secrets in **S** that satisfy the policy and the group key k. PI indirectly encodes the policy such that a user can use PI along with its secrets only if the user satisfies the policy used to generate PI.

- **KeyDer(s, PI)**: It takes the user's secret s and the public information PI to output the group key. The derived group key is equal to k if and only if $s \in$ **S** and satisfies the policy.

- **Update(S)**: Whenever the set **S** changes, a new group key k' is generated. Depending on the construction, it either executes the **KeyGen** algorithm again or incrementally updates the output of the last **KeyGen** algorithm.

3 Basic Approach to Privacy-Preserving ABAC

Using our AB-BGKM scheme, we have developed an ABAC mechanism whereby a user is able to decrypt the data if and only if its identity attributes satisfy the data owner's policies, whereas the data owner and the cloud learn nothing about user's identity attributes. The mechanism is fine-grained in that different policies

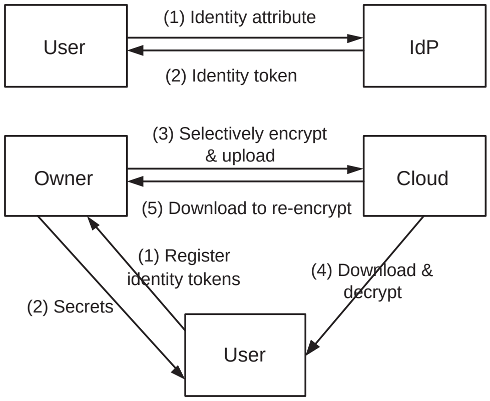

Figure 1: Overall System Architecture

can be associated with different sets of data items. A user can derive only the encryption keys associated with the sets of data items the user is entitled to access.

We now give an overview of the overall scheme. As shown in Figure 1, our scheme for policy based content sharing in the cloud involves four main entities: the *Data Owner* (Owner), the *Users* (Usrs), the *Identity Providers* (IdPs), and the *Cloud Storage Service* (Cloud). Our approach is based on three main phases: *identity token issuance*, *identity token registration*, and *data management*.

Identity Token Issuance

IdPs issue *identity tokens* for certified identity attributes to Usrs. An identity token is a Usr's identity encoded in a specified electronic format in which the involved identity attribute value is represented by a semantically secure cryptographic *commitment*.[2] We use the Pedersen commitment scheme [8]. Identity tokens are used by Usrs during the identity token registration phase.

Identity Token Registration

In order to be able to decrypt the data to be downloaded from the Cloud, Usrs have to register at the Owner. During the registration, each Usr presents its identity tokens and receives from the Owner a set of secrets for each identity attribute based on the **SecGen** algorithm of the AB-GKM scheme. These secrets are later used by Usrs to derive the keys to decrypt the sets of data items for which they satisfy the access control policy using the **KeyDer** algorithm of the AB-GKM scheme. The Owner delivers the secrets to the Usrs using a privacy-preserving approach based on the OCBE protocols [4]. The OCBE protocols ensure that a Usr can obtain the secrets if and only if the Usr's committed identity attribute value (within Usr's identity token) satisfies the matching condition in the Owner's access control policy, while the Owner learns nothing about the identity attribute value. Note that not only the Owner does not learn anything about the actual value of Usrs' identity attributes but it also does not learn which policy conditions are verified by which Usrs, thus the Owner cannot infer the values of Usrs' identity attributes.

Data Management

The Owner groups the access control policies into *policy configurations*. The data items are partitioned into sets of data items based on the access control policies. The Owner generates the keys based on the access control policies in each policy configuration using the **KeyGen** algorithm of the AB-GKM scheme and selectively encrypts the different data item sets. These encrypted data item sets are then uploaded to the Cloud. Usrs download encrypted data item sets from the Cloud. The **KeyDer** algorithm of the AB-GKM

[2] A cryptographic commitment allows a user to commit to a value while keeping it hidden and preserving the user's ability to reveal the committed value later.

scheme allows Usrs to derive the key K for a given policy configuration using their secrets in an efficient and secure manner. With this scheme, our approach efficiently handles new Usrs and revocations to provide forward and backward secrecy. The system design also ensures that access control policies can be flexibly updated and enforced by the Owner without changing any information given to Usrs.

An Example

Using the same EHR system presented earlier, we now provide an example showing the data management phase.

The data items of an EHR such as Contact Info, Lab Report and Clinical Record are allowed to be accessed by different employees based on their roles and other identity attributes. Suppose the roles for the hospital's employees are: receptionist (rec), cashier (cas), doctor (doc), nurse (nur), data analyst (dat), and pharmacist (pha). Three selected ACPs of the EHR system are shown below.

1. $ACP_1 =$ (“$\mathsf{role} =$ rec”, $\{\langle$ContactInfo$\rangle\}$)
2. $ACP_2 =$ (“$\mathsf{role} =$ doc”, $\{\langle$ClinicalRecord$\rangle\}$)
3. $ACP_3 =$ (“$\mathsf{role} =$ nur $\wedge$ $\mathsf{level} \geq 5$”, $\{\langle$ContactInfo$\rangle, \langle$ClinicalRecord$\rangle\}$)

The first ACP says that a receptionist can access Contact Info data items, the second says that a doctor can access Clinical Record data items, and the third says that a nurse with level greater than or equal to 5 can access Contact Info and Clinical Record data items. Based on these policies the hospital identifies the policy configuration for each set of data items. For the above sample policies, the hospital creates two policy configurations as follows:

1. $\mathsf{PC}_1 = (\langle$ContactInfo$\rangle$: ACP_1, ACP_3)
2. $\mathsf{PC}_2 = (\langle$ClinicalRecord$\rangle$: ACP_2, ACP_3)

The hospital generates a group key and public information using the **KeyGen** algorithm for PC_1, and encrypts Contact Info data items using the group key. Hospital employees who satisfy either ACP_1 or ACP_3 in PC_1 can derive the group using the **KeyDer** algorithm and decrypt the Contact Info data items downloaded from the cloud. A similar process is followed for PC_2 and other policy configurations not shown in the example.

3.1 Implementation and Security/Performance Analysis

We have implemented our basic approach on Amazon S3 which is a popular cloud based storage service. The content management consists of two tasks. First, the Owner encrypts the data item sets based on the access control policies and uploads the encrypted sets along with some meta-data. Then, authorized users download the encrypted data items sets and meta-data from the Cloud, and decrypt the data item sets using the secrets they have.

Now we illustrate the interactions of the Owner with Amazon S3 as the Cloud. In our implementation, we have used the REST API to communicate with Amazon S3. Figure 2 shows the overall involvement of the Owner in the user and content management process when uploading the data item sets to Amazon S3. While the fine-grained access control is enforced by encrypting using the keys generated through the AB-GKM scheme, it is important to limit the access to even the encrypted data item sets in order to minimize the bandwidth utilization. We associate a hash-based message authentication code (HMAC) with each encrypted data item sets such that only the users having valid identity attributes can produce matching HMACs.

Initially the Owner creates a *bucket*, which is a logical container in S3, to store encrypted data item sets as *objects*. Subsequently, the Owner executes the following steps:

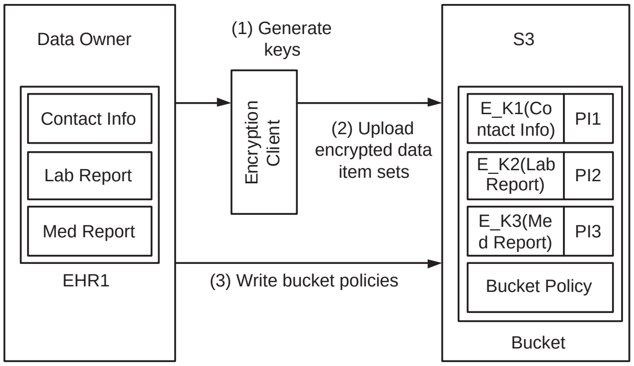

Figure 2: Overall involvement of the Owner

1. The Owner generates the symmetric keys using the AB-GKM's **KeyGen** algorithm and instantiates an encryption client. Note that the Owner generates a unique symmetric key for each policy configuration.

2. Using the encryption client as a proxy, the Owner encrypts the data item sets and uploads the encrypted data item sets along with the public information needed to generate the keys as the meta-data.

3. The Owner generates HMACs using the symmetric keys and PIs. These HMACs are used to write *bucket policies* that control access to encrypted data item sets. The bucket policies define access rights for the encrypted data item sets. It should be noted that only the Owner has access to the bucket policies. While S3 has access to the HMACs, by just knowing the HMACs S3 is not able to decrypt the data.

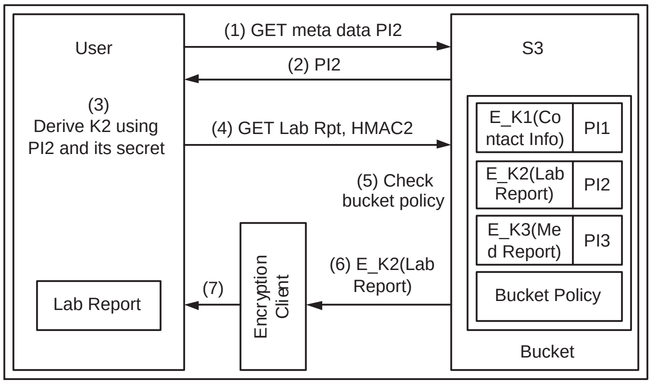

Figure 3: Downloading data item sets from Amazon S3 as the Cloud

Now we look at users' involvement in content management. Figure 3 shows the overall involvement of users with Amazon S3 when downloading encrypted sets of data items. The following steps are involved:

1. Users are allowed to download the meta-data for any encrypted data item set.

2. A user downloads the public information associated with the data item set it wants to access.

3. Using the **KeyDer** algorithm of the AB-GKM scheme, the user derives the symmetric key. Notice that the user can derive a valid symmetric only if its identity attributes satisfy the access control policy associated with the data item set.

4. Using the public information and the derived symmetric key, the user creates an HMAC and submits it to S3.

5. S3 checks if the HMAC in the bucket policy matches with the one the user submitted. If it does not match, it denies access.

6. The user downloads the encrypted data item set.

7. The encryption client decrypts the data item set.

3.2 Performance Enhancements

In this section, we discuss some enhancements to improve the performance of the basic ACV-BGKM scheme [9], which is the underlying construct of the AB-GKM scheme,by using two techniques: bucketization and subset cover.

3.2.1 Bucketization

Our ACV-BGKM scheme works efficiently even when there are thousands of Usrs. However, as the upper bound n of the number of involved Usrs gets large, solving the linear system $AY = 0$ over a large finite field $\mathbb{F}_q$, which the key operation in the **KeyGen** algorithm, becomes the most computationally expensive operation in our scheme. Solving this linear system with the method of Gaussian-Jordan elimination takes $O(n^3)$ time. Although this computation is executed at the data owner, which is usually capable of carrying on computationally expensive operations, when n is very large, e.g., $n = 100,000$, the resulting costs may be too high for the data owner. Due to the non-linear cost associated with solving a linear system, we can reduce the overall computational cost by breaking the linear system in to a set of smaller linear systems. We have thus defined a two-level approach. Under this approach, the data owner divides all the involved Usrs into multiple "buckets" (say m) of a suitable size (e.g., 1000 each), computes an intermediate key for each bucket by executing the **KeyGen** algorithm, and then computes the actual group key for all the Usrs by executing the **KeyGen** algorithm with the intermediate keys as the secrets. Note that the intermediate key generation can be parallelized as each bucket is independent. The data owner executes $m + 1$ **KeyGen** algorithms of smaller size. The complexity of the **KeyGen** algorithm is proportional to $O(n^3/m^2 + m^3)$. It can be shown that the optimal solution is achieved when m reaches close to $n^{3/5}$.

3.2.2 Subset Cover

The bucketization approach becomes inefficient when the bucket size increases. The issue is that the bucketization still utilizes the basic ACV-BGKM scheme. In our basic ACV-BGKM scheme, as each Usr is given a single secret, the complexity of PI and all algorithms is proportional to n, that is, the number of Usrs in the group. We have thus defined an approach based on a technique from previous research on broadcast encryption [3] to improve the complexity to sub-linear in n. Based on such technique, one can make the complexity sub-linear in the number of Usrs by giving more than one secret during **SecGen** for each attribute Usrs possess. The secrets given to each Usr overlaps with different subsets of Usrs. During the **KeyGen**, the data owner identifies the minimum number of subsets to which all the Usrs belong and uses one secret per the identified subset. During **KeyDer**, a Usr identifies the subset it belongs to and uses the corresponding secret to derive the group key. Group dynamics are handled by making some of the secrets given to Usrs invalid.

4 Two-layer Encryption Approach to Privacy-Preserving ABAC

Our basic approach follows the conventional data outsourcing scenario where the Owner enforces *all* the access control policies through selective encryption and uploads encrypted data to the untrusted Cloud. We refer to this approach as single layer encryption (SLE). The SLE approach supports fine-grained attribute-based access control policies and preserves the privacy of users from the Cloud. However, in such an approach, the Owner is in charge of encrypting the data before uploading it to the third-party server as well as re-encrypting the data whenever user credentials or authorization policies change and managing the encryption keys. The Owner has to download all affected data before before performing the selective encryption. The Owner thus incurs high communication and computation costs, which then negate the benefits of using a third party service. A better approach should delegate the enforcement of fine-grained access control to the Cloud, so to minimize the overhead at the Owner, whereas at the same time assuring data confidentiality from the third-party server.

In this section, we provide an overview of an approach, based on two layers of encryption, that addresses such requirement. Under such approach, referred to as *two-layer encryption* (TLE), the Owner performs a coarse grained encryption, whereas the Cloud performs a fine grained encryption on top of the data encrypted by the coarse grained encryption. A challenging issue in this approach is how to decompose the ABAC policies such that the two-layer encryption can be performed. In order to delegate as much access control enforcement as possible to the Cloud, one needs to decompose the ABAC policies so that the Owner only needs to manage the minimum number of attribute conditions in these policies that assures the confidentiality of data from the Cloud. Each policy should be decomposed into two subpolicies such that the conjunction of the two subpolicies result in the original policy. The two-layer encryption should be performed such that the Owner first encrypts the data based on one set of subpolicies and the Cloud re-encrypts the encrypted data using the other set of policies. The two encryptions together enforce the original policies as users should perform two decryptions in order to access the data. For example, consider the policy $(C_1 \wedge C_2) \vee (C_1 \wedge C_3)$. This policy can be decomposed as two subpolicies C_1 and $C_2 \vee C_3$. Notice that the decomposition is consistent; that is, $(C_1 \wedge C_2) \vee (C_1 \wedge C_3) = C_1 \wedge (C_2 \vee C_3)$. The Owner enforces the former by encrypting the data for the users satisfying the former and the Cloud enforces the latter by re-encrypting the Owner encrypted data for the users satisfying the latter. Since the Cloud does not handle C_1, it cannot decrypt the Owner encrypted data and thus confidentiality is preserved. Notice that users should satisfy the original policy to access the data by performing two decryptions. An analysis of this approach suggests that the problem of decomposing for coarse and fine grained encryption while assuring the confidentiality of data from the third party and the two encryptions together enforcing the policies is NP-complete. We have thus investigated optimization algorithms to construct near optimal solutions to this problem. Under our TLE approach, the third party server supports two services: the storage service, which stores encrypted data, and the access control service, which performs the fine grained encryption.

As shown in Figure 4, we utilize the same AB-GKM scheme that allows users whose attributes satisfy a certain policy to derive the group key and decrypt the content they are allowed to access from the Cloud. Our proposed approach assures the confidentiality of the data and preserves the privacy of users from the access control service as well as the cloud storage service while delegating as much of the access control enforcement as possible to the third party through the two- layer encryption technique.

The TLE approach has many advantages. When the policy or user dynamics changes, only the outer layer of the encryption needs to be updated. Since the outer layer encryption is performed at the third party, no data transmission is required between the Owner and the third party. Further, both the Owner and the third party service utilize the AB-GKM scheme for key management whereby the actual keys do not need to be distributed to the users. Instead, users are given one or more secrets which allow them to derive the actual symmetric keys for decrypting the data.

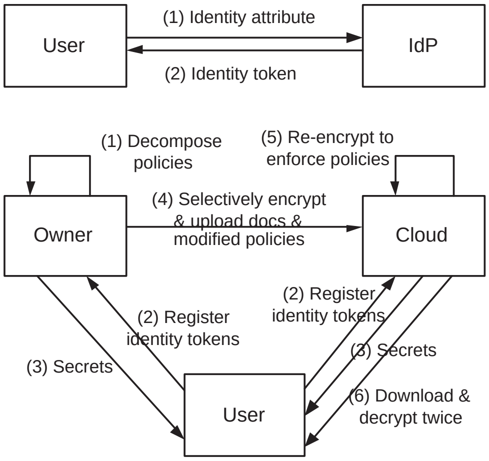
User
(1) Identity attribute
IdP
(2) Identity token
(1) Decompose policies
(5) Re-encrypt to enforce policies
Owner
(4) Selectively encrypt & upload docs & modified policies
Cloud
(2) Register identity tokens
(2) Register identity tokens
(3) Secrets
(3) Secrets
(6) Download & decrypt twice
User

Figure 4: Two Layer Encryption Approach

5 Comparison of Approaches

In this section we compare ABE-based existing approaches as a whole and the two AB-GKM based approaches presented earlier. A common characteristic of all these approaches is that they support secure attribute based group communication.

Table 1: Comparison of Approaches

Property	**ABE**	**SLE**	**TLE**
Cryptosystem	Asymmetric	Symmetric	Symmetric
Secure attribute based group communication	Yes	Yes	Yes
Efficient revocation	No	Yes	Yes
Delegation of access control	No	No	Yes

As shown in Table 1, while ABE-based approaches rely on asymmetric cryptography, our two approaches rely only on symmetric cryptography which is more efficient than the asymmetric cryptography. A key issue in the ABE-based approaches is that they do not support efficient user revocations unless they use additional attributes [2]. Our schemes address the revocation issue. It should be noted that the ABE based approaches and our SLE approach follows the conventional data outsourcing scenario by which the data owner manages all users and data before uploading the encrypted data to the cloud, whereas the TLE based approach provides the advantage of partial management of users and data in the cloud itself while assuring confidentiality of the data and privacy of users. With ever increasing user base and large amount of data, while such delegation of user management and access control is becoming very important, it also has trade offs in terms of privacy. Compared to the SLE approach, in the TLE approach, the data owner has to reveal partial access control policies to the cloud which may allow the cloud to infer some details about the identity attributes of users. It is an interesting topic to investigate how to construct symmetric key based practical solutions to hide the access control policies from the cloud while utilizing the benefits of delegation of control.

6 Conclusions

Current trends in computing infrastructures like Service Oriented Architectures (SOAs) and cloud computing are further pushing publishing functions to third-party providers to achieve economies of scale. However, recent surveys by IEEE and Cloud Security Alliance (CSA) have found that one of the key resistance factor for companies and institutions to move to the cloud is represented by data privacy and security concerns. Our AB-GKM based approaches address such privacy and security concerns in the context of efficient and flexible sharing and management of sensitive content. Compared to state of the art ABE based approaches, our approaches support efficient revocation and management of users which is a key requirement to construct scalable solutions.

References

[1] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *SP 2007: Proceedings of the 28th IEEE Symposium on Security and Privacy*, pages 321–334, 2007.

[2] J. Camenisch, M. Dubovitskaya, R. R. Enderlein, and G. Neven. Oblivious transfer with hidden access control from attribute-based encryption. In *SCN 2012: Proceedings of the 8th International Conference on Security and Cryptography for Networks*, pages 559–579, 2012.

[3] D. Halevy and A. Shamir. The LSD broadcast encryption scheme. In *CRYPTO 2001: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 47–60, 2002.

[4] J. Li and N. Li. OACerts: Oblivious attribute certificates. *IEEE Transactions on Dependable and Secure Computing*, 3(4):340–352, 2006.

[5] M. Nabeel and E. Bertino. Towards attribute based group key management. In *CCS 2011: Proceedings of the 18th ACM conference on Computer and communications security*, 2011.

[6] M. Nabeel, N. Shang, and E. Bertino. Privacy preserving policy based content sharing in public clouds. *IEEE Transactions on Knowledge and Data Engineering*, 99, 2012.

[7] OpenID. `http://openid.net/` [Last accessed: Oct. 14, 2012].

[8] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO 1991: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 129–140, 1992.

[9] N. Shang, M. Nabeel, F. Paci, and E. Bertino. A privacy-preserving approach to policy-based content dissemination. In *ICDE 2010: Proceedings of the 2010 IEEE 26th International Conference on Data Engineering*, 2010.

[10] S. Yu, C. Wang, K. Ren, and W. Lou. Attribute based data sharing with attribute revocation. In *ASIACCS 2010: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 261–270, 2010.

[11] S. Yu, C. Wang, K. Ren, and W. Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *INFOCOM 2010: Proceedings of the 29th conference on Information communications*, pages 534–542, 2010.