A PRACTICAL CLIENT APPLICATION BASED ON ATTRIBUTE-BASED ACCESS CONTROL FOR UNTRUSTED CLOUD STORAGE

Julian Jang-Jaccard

Computer Science and Information Technology, Institute of Natural and Mathematical Sciences (INMS), Massey University, New Zealand

ABSTRACT

One of widely used cryptographic primitives for the cloud application is Attribute Based Encryption (ABE) where users can have their own attributes and a ciphertext encrypted by an access policy. Though ABE provides many benefits, the novelty often only exists in an academic world and it is often difficult to find a practical use of ABE for a real application. In this paper, we discuss the design and implementation of a cloud storage client application which supports the concept of ABE. Our proposed client provides an effective access control mechanism where it allows different types of access policy to be defined thus allowing large datasets to be shared by multiple users. Using different access policy, each user only needs to access only a small part of the big data. The goal of our experiment is to explore the right set of strategies for developing a practical ABE-based system. Through the implementation and evaluation, we have determined the various characteristics and issues associated with developing a practical ABE-based application.

KEYWORDS

Attribute-based encryption, cloud storage service, Access control

1. INTRODUCTION

Cloud storages are widely used by organizations and people to outsource data. They have become more popular since cloud service providers offer their users cost efficient data storage with high mobility and reliability. However, the data in the cloud can be misused by cloud service providers or leaks the private sensitive data when there are inappropriate security mechanisms in place as seen in [1], [2]. Securing the data on the cloud still remains as a primary concern for many users.

One of the most promising ways to protect data on the cloud is encryption. Encryption offers additional protection as data is encrypted before they are uploaded to the cloud and decrypted after they are downloaded. However, traditional encryption technique, such as public key encryption, only allows a single user to decrypt the data which is too restrictive and ineffective in many cases. In particular in many modern applications, the data size is big and the data often

David C. Wyld et al. (Eds) : CNSA, DMDB, AIAP - 2018
pp. 01– 15, 2018. © CS & IT-CSCP 2018 DOI : 10.5121/csit.2018.80101

requires many different types of access policies to allow multiple users to share the data while each user only have access to a small part of it.

To improve the data sharing, cloud applications require more flexible fine-grained access control to support multiple users with different access requirements. Recent achievements in cryptographic theory such as Attribute Based Encryption (ABE) [15], [16] provide solutions to this data sharing dilemma. In ABE, in particular Ciphertext-Policy ABE (CP-ABE), users have their own attributes and a ciphertext is created with association of an access policy. Multiple users are able to decrypt the ciphertext if the users' attributes pass through the ciphertext's access structure.

Though the novelty provided by ABE scheme has been endorsed, the novelty only exists in an academic world as theory. The practical use of ABE scheme in real life applications have not been explored well. In this paper, we discuss the design and implementation details to apply an ABE scheme for a real life application. Our proposed client application sits in between the cloud application (such as DropBox, Google Drive, Microsoft OneDrive) and cloud storages (such as Amazon S3). Our client application not only provides encryption to safeguard user's sensitive data but also offer an effective access control mechanism to allow multiple users to share the data. The goal of our proposal is to provide opportunities to explore the right set of strategies for developing a practical ABE-based application. We offer the details of various algorithm implementations and issues associated with developing a practical ABE-based application.

The paper is organised as following: design considerations are presented in Section 2. Section 3 provides background material our proposal is based on. Section 4 describes the system architecture. Our implementation details, algorithms, and performance results are described in Section 5. The lessons we learnt during the proof-of-concept demonstrator is presented in Section 6. Section 7 concludes the paper along with the future work.

2. DESIGN CONSIDERATION

In this section, we discuss a number of priorities that we considered to design and implement our proposed client application.

2.1 Man-in-the-middle Attack

Most cloud storage service today provides a cloud application where a user can upload and subsequently download data using their own device at the comfort of home. More often than not, the data being transferred from user's device to the cloud storage server in not encrypted. This increases the chance for a malicious man-in-the-middle either to make an unauthorized access to user's data (i.e., breaking data confidentiality) or an unauthorized modification to the data (i.e., breaking data integrity). To provide data confidentiality and integrity, a protection mechanism must be employed such as an encryption. The secret key used for the encryption must be securely protected to prevent any potential snooping or stealth.

2.2 Dishonest Cloud Provider

Dishonest cloud provider makes an unauthorized attempt to read user's data. The cloud storage provider has all necessary tools and mechanisms to access the data that is under its full control.

Either by a malicious code implanted in the cloud server or by deliberate attempts by a dishonest cloud server administrator, the chance for data leakage increases if user's data is improperly protected. In addition, it is also possible that dishonest cloud provider makes copies of user's data. It is a common practice for cloud storage providers to make copies of original data and store them in extra storages, such as in the backup media or replicated databases. These copied datasets are then re-applied when an unexpected disaster occurs therefore user's data is recovered. However, even after user's subscription expires, these copies of the original data may still remain somewhere in the cloud server and targeted for further compromise. To prevent such misuse, data much be encrypted such a way that it does not reveal its original content other than to authorized users.

2.3 Data Sharing

One of the most widely used solutions to protect the data is encryption. In public key encryption which is one of the most widely used encryption scheme, a user encrypts data before uploading it to the cloud and decrypts the data after it is downloaded. However, such traditional encryption techniques do not support multi-party collaboration without re-distributing keys that are used for the encryption which creates a serious key management problem. In addition, as many modern applications produce big data sets that are large in size, it is often inefficient to download and upload the whole dataset.

The recent advancement of Attribute-Based Encryption (ABE) provides a solution to the issues associated with traditional encryption technique by supporting more fine-grained access control mechanism. The offer of such fine-grained access control is better suited when dealing with big data that are accessed by multiple users.

2.4 Access Policy Expression

Traditional public-key encryption requires that decryption to be done by one particular user and does not allow more complex access control policies. ABE relaxes such limitation by allowing decryption to be decided by an access policy tree. However, up until recently, the predicate expression used to define access polices were limited to be monotone which consists of AND, OR, or threshold gates. This does not allow the representation of negative constraints which raises a problem in scenarios where conflicts of interest naturally arise. For example, if Bob is not allowed to see a sensitive document shared by his colleges Alice, Sara and Kevin, it is more intuitive to define an access policy with ‘NOT Bob allowed’ than ‘allow Alice’ AND ‘allow Sara’ AND ‘allow Kevin’. The CP-ABE we apply has the strength offering the NOT gate in the predicate hence the name non monotonic.

3. BACKGROUND

Our work utilizes non-monotonic Ciphertext-Policy Attribute Based Encryption (CP-ABE) scheme originally proposed by Yamada et al. [12]. The non-monotonic scheme can be summarised as following four algorithms

- **Setup** (λ). It chooses a bilinear group $(\mathbb{G}, \mathbb{G}_T)$ of prime order $p > 2^\lambda$ with $g \in \mathbb{G}$. Then it chooses random exponents $\alpha, \beta \in \mathbb{Z}_p$ and $H, U, V, W \in \mathbb{G}$. Set $V' = U^b$. It generates a public key **mpk** and a master secrete key **msk**. **mpk** = $(g, H, U, V, V', W, e(g,g)^\alpha$ and **msk** = (α, β)

- **Encrypt(mpk, M, T)**. The encryption algorithm encrypts a message **M** under a non-monotonic access structure T over a set of attributes that are associated with a linear secret sharing scheme (L, π) using the master public key **mpk**. L is an $l \times m$ matrix. To produce a cipertext C, first it picks a random $\mathbf{s} = (s, s_1, \ldots s_m) \in \mathbb{Z}_p^m$, computes share of s for $\pi(i)$ by $\lambda_i = \langle \mathbf{L}_i \cdot \mathbf{s} \rangle$ for $i = 1, \ldots l$, and then computes $C_0 = \mathbf{M} \cdot e(g,g)^{\alpha \cdot s}$, $C_1 = g^s$. It also computes $(C_{i,1}, C_{i,2}, C_{i,3})$ for every $i = 1, \ldots l$ as follows.
$$\begin{cases} C_{i,1} = W^{\lambda_i} V^{t_i}, & C_{i,2} = (U^{x_i} H)^{-t_i}, C_{i,3} = g^{t_i} \text{ if } \pi(i) = x_i \\ C_{i,1} = W^{\lambda_i} (V')^{t_i}, & C_{i,2} = (U^{x_i} H)^{-t_i}, C_{i,3} = g^{t_i} \text{ if } \pi(i) = x'_i \end{cases}$$
where $t_i \in \mathbb{Z}_p$. It outputs the ciphertext $C = (C_0, C_1, \{C_{i,1}, C_{i,2}, C_{i,3}\}_{i \in [l]})$.

- **KeyGen(msk, mpk, w)**. The key generation algorithm takes a set of attribute $w = \{x_1, \ldots x_k\} \subset \mathbb{Z}_p$ and outputs a key that can identify the set (of attributes). It first chooses random $r, r_1, \ldots r_k \in \mathbb{Z}_p$ and $r'_1, \ldots r'_k \in \mathbb{Z}_p$ such that $r'_1 + \cdots + r'_k = r$. The private key **sk** is generated as;
$$\mathbf{sk} = \left(D_1 = g^\alpha W^r, D_2 = g^r, \begin{Bmatrix} k_{i,1} = V^{-r} (U^{w_i} H)^{r^i}, k_{i,2} = g^{r_i} \\ k'_{i,1} = (U^{bw_i} H^b)^{r'^i}, k'_{i,2} = g^{br'_i} \end{Bmatrix}_{i \in [k]} \right).$$

- **Decrypt(mpk, C, w, sk)**: We assume that access structure T is satisfied y the attribute set w which means decryption is possible. Let I = $\{i | \pi(i) \in w'\}$. Because $w'$is authorized in the access tree T, the recipient can efficiently computer the reconstruction coefficients $\{(i, \mu_i)\}_{i \in I}$=Recon $_{L,\pi}(w')$ such that $\sum_{i \in I} \mu_i \lambda_i = s$. Then it parses $C = (C_0, C_1, \{C_{i,1}, C_{i,2}, C_{i,3}\}_{i \in [l]})$, $sk = (D_1, D_2, \{K_{i,1}, K_{i,2}, K'_{i,1}, K'_{i,2}\}_{i \in [k]})$ and computes $e(g,g)^{r \cdot \lambda_i}$ for each $i \in I$ as follows;
$$\begin{cases} e(C_{i,1}, D_2) \cdot e(C_{i,2}, K_{\tau,2}) \to e(g,W)^{r\lambda_i} \text{ if } \pi(i) = x_i \\ e(C_{i,1}, D_2) \cdot \prod_{j \in [k]} (e(C_{i,3}, K'_{j,1}) \cdot e(C_{i,2}, K'_{j,2}))^{\frac{1}{x_i - w_j}} = e(g,W)^{r\lambda_i}, \text{ if } \pi(i) = x'_i \end{cases}$$
Here τ indicates the index such that $w_\tau = x_i$. Such τ exists if $i \in I$ and $\pi(i)$ is non-negated attribute. With all that, it computes $e(C_1, D_2) \cdot \prod_{i \in I} (e(g,W)^{r\lambda_i})^{-\mu_i} = e(g^s, g^\alpha) e(g,W)^{sr} e(g,W)^{-r \sum_{i \in I} \mu_i \lambda_i} = e(g,g)^{\alpha \cdot \beta}$. Finally it recovers the message by $c_0 / e(g,g)^{s\alpha} = \mathbf{M}$.

4. SYSTEM ARCHITECTURE

4.1 System Overview

Our client application is based on the model proposed in [8]. The main purpose of the client application is to provide an encryption scheme with flexible access policies. Figure 1 illustrates the overview of our proposed client application.

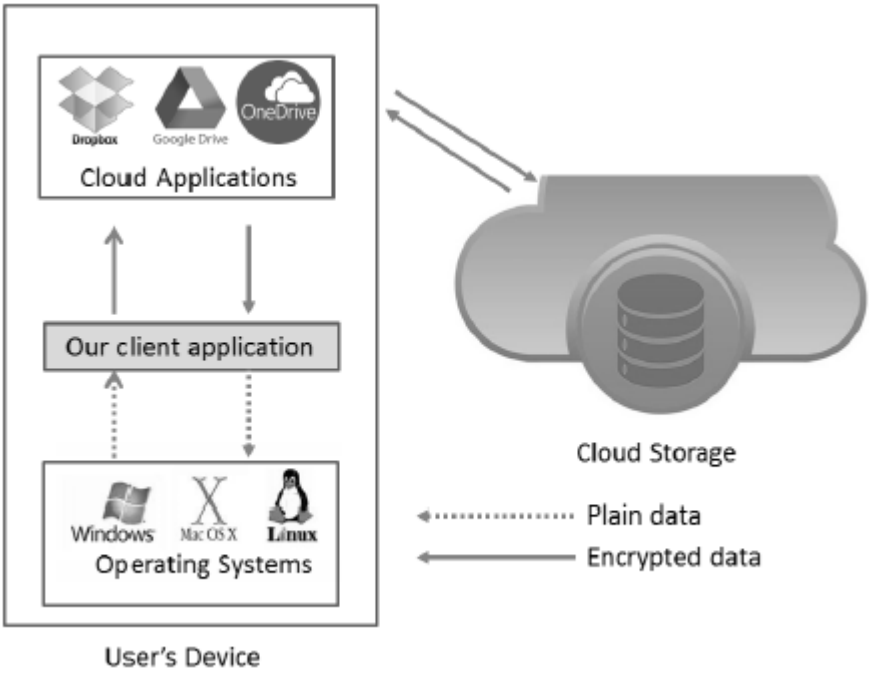


Figure 1: system overview

Our proposed client application sits between the operating system (e.g., Windows, Linux) and cloud storage application (e.g., DropBox, Google Drive, Microsoft OneDrive) installed in user's device (e.g., PCs, laptop, tables, or smartphones). Our client application provides an extra layer of protection to safeguard user's data and support attribute-based access control mechanism to support data sharing. Our client application can be seamlessly integrated with any cloud applications which we presume that it is pluggable from one application over the other.

Our application interacts with users. Here, the term users include the authorized devices, such as PCs, laptop, tables, or smartphones, laptop owned by individuals. Users can be categorised as encryptors and recipients. An encryptor refers to a type of user who owns the data and has full control of deciding whom it wants to share the data with. The encryptor can specify a set of access policies over the data to enforce access rules to control the access permission. The recipient refers to a type of user who wants to share the data with the encryptor. The recipient must provide a right set of attributes to proof that he/she satisfies the access rules imposed on the shared data to pass the permit.

Users contact administrator which can just be another user. The major role of the administrator is to generate master public and private key pairs under non-monotonic CP-ABE scheme. The administrator publishes the public keys. It also generates private keys based on user's attributes for all users. Once private keys are generated, they are distributed to corresponding users. We do not cover the details of private key distribution in this paper other than assuming that the private keys are delivered using a secure channel such as IPSec or SSL.

The cloud storage is a remote storage facility and is typically provided by cloud storage providers. The user can use cloud storage applications to store files in the cloud to share with others. Cloud storage providers may support security mechanisms to protect user's data from

potential data loss and from unauthorized access. However, the access control mechanism provided by our client application does not depend on the underlying supports and does not share information regarding the encryption and decryption strategies and parameters.

4.2 System Operations

Figure 2 describes the details of the operations in terms of message exchange among various system components.

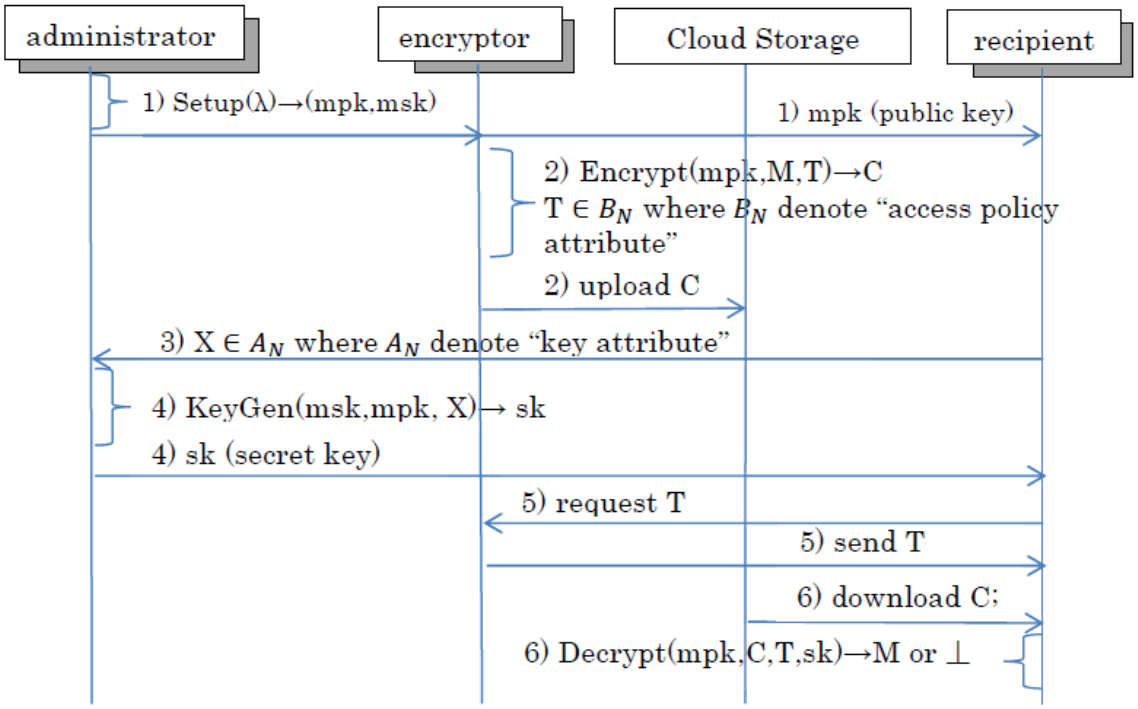


Figure 2: System operations

Note that our client application runs on the administrator, encryptor and the recipient as depicted by a gray box behind them and assists the following operations.

1) The operation here is a set up by an administrator. The client application running on the administrator takes a security parameter λ, in our case, it is a random number generated by a random number generator provided by a java cryptographic package our client application utilises. The result of the setup is a key-pair, one for a public key mpk and a corresponding private key msk.

2) In the meantime, a user acting as an encryptor wish to upload a file (M) to a cloud storage. The encryptor uses the client application running in the encrytor's device to encrypt the file. At this stage, the encryptor also defines an access policy (T) . A typical example of an access policy tree is depicted in Figure 3.

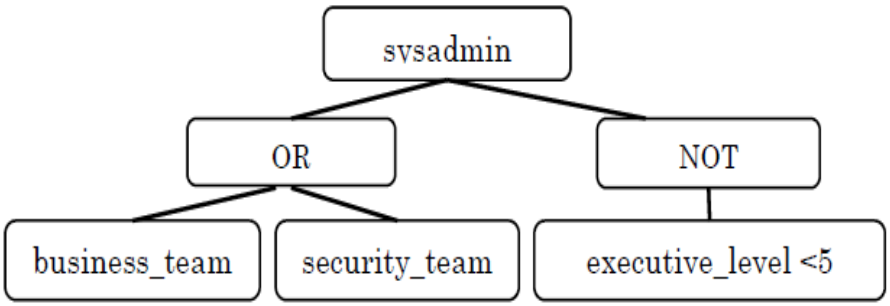


Figure 3: an example of an access policy tree

which can be defined as a simple Boolean formula;

Y = (sysadmin AND (“business_team” OR “security_team) AND
NOT (executive_level <5))

The access policy is associated with the encrypted data then uploaded to the cloud application which in turn subsequently sends the data to the cloud storage over the Internet.

3) Now let’s assume that there is a recipient who wants to download a portion of the data uploaded by the encryptor to share. To proceed, the recipient first needs to send his/her attributes (e.g., job title, department, and executive level) to an administrator. Let’s call the recipient’s attribute X which can be defined as;

X = (job_title = sysadmin, department = security_team, executive_level = 7)

4) The administrator generates a private key sk according to the attributes then sends it to the corresponding recipient.

5) In the meantime, the recipient also sends a request to send an access policy T that is associated with the data the recipient wants to share with the encryptor.

6) Using the private key sk and access policy T received from the administrator and the encryptor, the recipient downloads the data and attempts the decryption. If the attributes associated with recipient’s private key satisfies with the predicate defined in the access policy, the data is decrypted M. Nothing is returned $\perp$ if the key does not satisfy.

5. SYSTEM IMPLEMENTATION

5.1 Libraries

The CP-ABE scheme we use in our client application is a class of pairing based cryptography. It applies each encryptor’s access rules as the attribute in the access policy tree. The public and private key pair is related to the attributes while the ciphertext is associated with an access policy. Our implementation is split into two packages:

cpabe: is a package that implements the CP-ABE scheme introduced by [12].

App: is a package that implements the higher level functions such as user interface, interpreting access policy and store and retrieve the data from the cloud.

The cpabe package uses java based pairing cryptography JPBC library [13]. JPBC is a java based open library that supports cryptographic methods. JPBC is a port of the Pairing-Based Cryptography Library (PBC) developed by Ben Lynn [14] to perform the mathematical operations underlying pairing-based cryptosystems directly in Java. Our current client application integrates with Aamzon S3 which represents a cloud application that runs along with our client application. We use the official Amazon Java SDK for API 1.11.58 [17] to integrate Amazon S3 into our client using Java.

5.2 Essential Algorithms

Attribute/Access Policy Parser: In CP-ABE scheme, each user's private key is associated with a set of attributes which represent their capabilities. A ciphertext is encrypted such that only users whose attributes satisfy a certain policy can decrypt. To use the scheme, we first need a scheme to define attributes and a parse an access policy which can understand them. CP-ABE scheme we implement basically support any Boolean formula, but interpreting unstructured Boolean formula and computing parameters using Linear Secret Sharing (LSS) scheme [18]. To parse an access policy, we allow encryptor to input an access policy under the following rules;

1. On a line only AND and NOT gate can be used
2. Split a line means OR gate

"Att1" AND "Att2" AND NOT "Att3" "Att1" AND "Att5"

Theses inputs enforce user to input an access policy as a disjunctive normal form and allows App package to parse the policy more efficiently.

System initiation: the set up phase is done by administrator without the user having to take any action. Under the hood, administrator runs the cpabe package to generate a public key and a master secret key. The input function and the outputs are defined as following.

INPUT: cpabe.setup()
OUTPUT: master_key pub_key

After running cpabe.setup() function, the administrator obtains a master_key to be used to produce private keys for recipients. The public key pub_key can be shared in the Cloud or sent to any users.

Secrete Key Generation: administrator uses this function to create private keys for all recipients who wishes to share the data encrypted by an encryptor. The function keygen() integrates the input attributes (i.e., each recipient's capabilities) in the private key generation using the public key and the master secret key generated in the setup() phase. The output is a private key

corresponding to a recipient. Attributes can come in a natural language but special characters cannot be included in the attributes.

INPUT: Cpabe.keygen(pub_key, master_key, attr)
OUTPUT: pri_key

Let's assume that there are two recipients Sara and Kevin at a company and the administrator wants to produce private keys. The private keys include the both hires' capabilities so that not all company documents can be decrypted by them. Sara's capability is defined as her job title is a sysadmin at IT department. Her office number is at 1431 and she was hired last year - let's say the date is denoted as yearX. In contrast, Kevin's capability is defined as his job title is a business staff at strategy team. He has the executive level 7 and sits in the office number 2362. He was hired 5 years ago – let's say the date is denoted as yearY. Sara's and Kevin's respective input parameters for the function keygen() follows.

Sara's private key generation:
INPUT: Cpabe.keygen(pub_key, master_key, "Sara sysadmin it_department office_1432 hire_yearX)
OUTPUT: sara_pri_key

Kevin's private key generation:
INPUT: Cpabe.keygen(pub_key, master_key, "Kevin business strategy_tem executive_level_7 office_2362 hire_yearY)
OUTPUT: Kevin_pri_key

The keygen() function allows some attributes are assigned a value while others a key simply "has" without further qualification. The date command can be used to help use the current time as an attribute value.

Encryption: encryptor uses this function to encrypt a file. Now assume that a staff member Bob in the company wants to encrypt a sensitive document. Bob only wish to share it with; someone who works in the sysadmin role in the security tem, or someone who is a business staff and either in the audit group or strategy team. Bob only needs the public key then can use the cpabe.enc to encrypt it with the access policy.

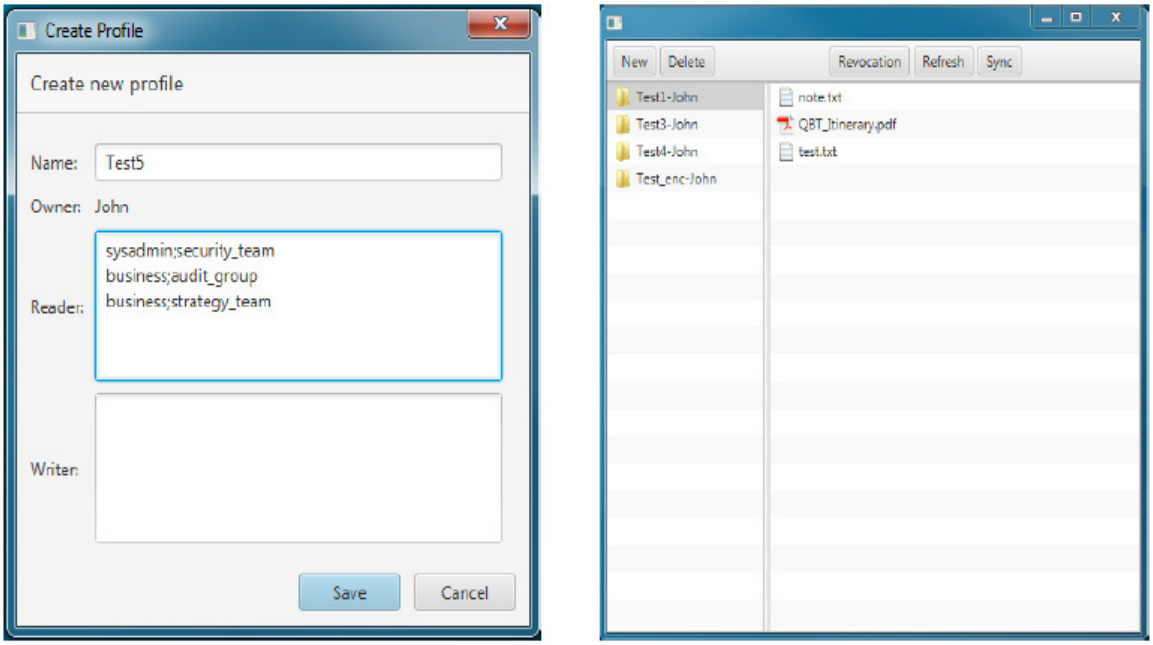
INPUT: Cpabe.enc(pub_key, security_report.pdf, "(sysadmin AND security_team) OR (business AND audit_group) OR (business AND strategy_team)"
OUTPUT: security_report.pdf.cpabe

Decryption: the recipients use this function to decrypt the file. Kevin can successfully decrypt the encrypted security report using his private key associated with his capabilities which satisfy the access policy associated with the encrypted document. Sara won't because the attributes of Sara's key does not satisfy the access policy.

INPUT: Cpabe.dec (pub_key, kevin_priv_key, security_report.pdf.cpabe)
OUTPUT: security_report.pdf

5.3 User Interface

In this section, we demonstrate a set of GUI screens we developed for our client desktop demonstrator which allows the users to manage and control access control mechanism based on the non-monotonic CB-ABE.



(a) Setting a profile for a secure folder (b) A secure folder for Enc/Decryption

Figure 4: client application GUI user interface

Figure 4 (a) shows the user interface screen that allows the encryptor to create a secure folder by setting an access policy that will commonly apply to the files in the folder. Reader box takes as input an access policy. For the simplicity, we use " ; " to denote AND gate and " ` " to denote NOT gate. Splitting line means OR gate. A user can create this secure folder in the Cloud (Amazon S3) by clicking the "Save" button.

Figure 4 (b) shows the screen that a list of secure folders and files in the selected folder which is synchronized with (Amazon S3). This window allows the recipients to perform an encryption/decryption operation. Encryption is performed by dragging a file in a file explorer or desktop to the right side of screen where a list of files are displayed. Then, the file is encrypted based on the access policy of the secure folder set when it created. Then, the encrypted file is uploaded and stored to the Cloud. For the decryption, a user simply clicks the file. As long as recipient's private key attributes matches with the access policy associated with the encrypted file, the file is downloaded from the Cloud and decrypted and stored the local computer.

While the processes, a public key and a corresponding recipient's private key are stored in an application folder in the local computer. Therefore, the Cloud cannot decrypt the files on their storage.

5.4 Implementation Results

We tested our implementation in the following system to measure and estimate the performance. For the elliptic curve, we use the type A (a_181_603) for the implementation.

Processor	Intel® Core™ i7-4600U CPU @ 2.10GHz 2.70 GHz
RAM	16 GB
OS	Windows 7 SP1 64-bit

For the fast encryption/decryption, we utilize the symmetric algorithm which is AES-256 to encrypt/decrypt files and hide the secret AES-128 key of each secure folder using CP-ABE scheme.

We estimate the times to execute each algorithm. The setup algorithm takes 94.8 ms on average. Also, KeyGen algorithm linearly increases for the number of attributes that a user has as shown in Table 1.

Table 1: performance of keygen depending on the number of user's attributes

# of attributes	2	4	6	8	10
KeyGen (ms)	230.2	425	616.3	816.3	1038.6

We also run encryption and decryption algorithm. The execution times are depending on both the number of attributes (non-negated attributes) and the number of attributes with not gate (negated attributes). We set a user to have five attributes but change the size of attributes for the simulations. The execution time of encryption increase almost evenly regardless of the type of attribute in an access policy. However, the execution time of decryption depends on the type of attributes in a policy. Negated attributes requires more decryption time.

# of non-negated attributes	1	2	3	4	5
Encryption (ms)	81	140.3	211.3	267.3	335.6
Decryption (ms)	38.4	60.7	82.9	102	130.4

# of negated attributes with 5 non-negated attributes	1	2	3	4
Encryption (ms)	401.6	436.4	503.8	577.8
Decryption (ms)	217	288.3	377.8	475.1

The time measures above does not depend on the size of a file since the file in the secure folder in the Cloud actually encrypted using AES-256 algorithm and CP-ABE was used only to encrypt/decrypt the AES secret key which is always 256 bits.

6. LESSONS LEARNED

ABE-based access control mechanism provides a great opportunity to better manage and share large datasets among multiple users. Though the promise is great, developing an ABE-based

implementation strategy is often difficult and time consuming due to lack of language and tools support.

The time of our implementation, there were only two reference implementations of monotonic CP-ABE-scheme, one using a C language and the other with Java. If any developer wants to implement a CP-ABE in other languages such as .NET or Python, they are left with no other option but having to implement all functionalities from scratch. Currently there is no available non-monotonic CP-ABE but ours.

Another problem with current support is with platform dependency. Though Java is independent from platform in theory but in practice it is often platform dependent. For example, Java crypto API we used for our application did not guarantee the same operation on different platforms. We used AES CBC algorithm for encryption/decryption. Though the encryption successfully worked on both desktop application and Android, the decryption did not work on Android. As it turned out, the way padding was added for the AES CBC algorithms were different from the desktop application to Android. The strategy adopted for AES CBC padding for Android was not compatible with cpabe library we were using; as a result, decryption operation didn't work on the Android application.

7. RELATED WORK

Cryptography is one of the most promising ways to providing secure access control [7] [11]. The initial applications of cryptographically enforced access control [3] [6] were influential, but omitted some details which are required to practical implementations such as access policy update and key distributions. Recently, more practical cryptographically enforced access control mechanisms were proposed. They support a fine-grained access control using an advanced cryptographic primitives such as ABE and Predicate Encryption (PE).

Li et al. [9] introduced a novel implementation of Muti-Authority ABE to provide an access control on the personal health records on the cloud storage. Their scheme supports a user revocation by redistributing users' private keys and update an access policy in ciphertexts. GORAM and A-GORAM [10] are suggested to provide a secure data sharing on the untrusted cloud storage. They provide a fine-grained access control using PE and hiding even an access patterns from the server. Their systems allow updating policy by re-encrypt both access policy and their corresponding data. Wang et al. [11] introduced Sieve which can provide a fine-grained access control on the untrusted cloud storage. Sieve supports dynamic and efficient user revocation. It needs to re-encrypt policy, but reduce a burden of re-encrypting all data using key homomorphic encryption [4][5]. Garrison et al. [7] showed that role based access control (RBAC$_0$) can be cryptographically enforced. Their system uses a simple identity based encryption or a traditional RSA, but they showed that this can be extended to support more complicated access control models using advanced cryptographic primitives such as HIBE and ABE.

Although those systems well realize complicated and practical access model with advanced cryptographic primitives, supporting dynamic access control is still difficult. Even very recent work [7][11] requires both redistributing all valid users' keys and updating ciphertexts to revoke invalid users. Redistributing all users' keys needs secure communications with users and

administrator and Updating ciphertexts requires that decryption and re-encryption. Therefore, they need intensive communications and computations.

Kim and Surya [8] suggested the system which provided a flexible revocation. Their system utilizes CP-ABE supports non-monotonic access structure suggested by Yamada et al. [12] and introduced a revocation algorithm without redistributing users' keys using negated attributes in access policies. However, their system is not system-wise implementation. It only implements and estimate the Yamada's ABE schemes in C using PBC [14].

8. CONCLUSION

We proposed a client application that implements a non-monotonic CP-ABE. Our proposed client application sits in between the cloud application and cloud storages offering not only encryption and decryption processing to safeguard user's sensitive data but also effective access control mechanism to allow multiple users sharing the data. We described the design consideration, system architecture and practical algorithms to apply an ABE-based scheme. We showed that it is possible to develop a ABE-based solution and can offer much flexible access control mechanisms for multiple users unlike public key infrastructure.

In a practical system, access policy is dynamic rather than static. Users can be added and deleted while the system is operating. Particularly, user revocation is difficult since invalid users must be successfully and immediately revoked from the system. We plan to implement a revocation mechanism [8] in our application in the near future.

ACKNOWLEDGEMENTS

The author would like to thank Jongkil Kim, CSIRO Australia, for providing the insights of the model this paper is based on and providing the implementation strategies. The author also thanks to Yao Leon Li for his assistance with the implementation and comments on the technical feedback

REFERENCES

[1] Operation Aurora. https://en.wikipedia.org/wiki/OperationAurora. Accessed: Dec. 2016.

[2] Tim Ring. Cloud computing hit by celebgate. 2015. https://www.scmagazineuk.com/cloud-computing-hit-by-celebgate/article/540448/. Accessed: Aug. 2017

[3] Selim G. Akl and Peter D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. ACM Trans. Comput. Syst., 1(3):239–248, 1983.

[4] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, EUROCRYPT, volume 8441 of LNCS, pages 533–556. Springer, 2014.

[5] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan.Key homomorphic prfs and their applications. In Ran Canetti and Juan A. Garay, editors, CRYPTO, volume 8042 of LNCS, pages 410–428. Springer, 2013.

[6] E. Gudes. The design of a cryptography based secure file system. IEEE Transactions on Software Engineering, SE-6(5):411–420, Sept 1980.

[7] William C. Garrison III, Adam Shull, Steven Myers, and Adam J.Lee. On the practicality of cryptographically enforcing dynamic access control policies in the cloud. In IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016, pages 819–838. IEEE Computer Society, 2016.

[8] Jongkil Kim and Surya Nepal. A cryptographically enforced access control with a flexible user revocation on untrusted cloud storage. Data Science and Engineering, 1(3):149–160, 2016.

[9] Ming Li, Shucheng Yu, Yao Zheng, Kui Ren, and Wenjing Lou. Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. IEEE Trans. Parallel Distrib. Syst., 24(1):131–143, 2013.

[10] Matteo Maffei, Giulio Malavolta, Manuel Reinert, and Dominique Schr¨oder. Privacy and access control for outsourced personal records.In IEEE Symposium on Security and Privacy, pages 341–358. IEEE Computer Society, 2015.

[11] Frank Wang, James Mickens, Nickolai Zeldovich, and Vinod Vaikuntanathan.Sieve: Cryptographically enforced access control for user data in untrusted clouds. In NSDI, pages 611–626, Santa Clara, CA, March 2016. USENIX Association.

[12] Shota Yamada, Nuttapong Attrapadung, Goichiro Hanaoka, and Noboru Kunihiro. A framework and compact constructions for non-monotonic attribute-based encryption. In Hugo Krawczyk, editor, PKC, volume 8383 of LNCS, pages 275–292. Springer, 2014.

[13] Angelo De Caro and Vincenzo Iovino. jpbc: Java pairing based cryptography. In Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011, pages 850–855, Kerkyra, Corfu, Greece, June 28 - July 1, 2011.

[14] Lynn B (2007) On the implementation of pairing-based cryptosystems, Ph.D. thesis, Ph.D. thesis, Stanford University.

[15] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, ACM Conference on Computer and Communications Security, pages 89–98. ACM, 2006

[16] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In EUROCRYPT, pages 457–473, 2005.

[17] AWS SDK for Java 1.11.58. https://aws.amazone.com/releasenotes/2482881671102750. Accessed: Aug. 2017

[18] Amos Beimel. Secure Schemes for Secret Sharing and Key Distribution. PhD thesis, Israel Institute of Technology, Technion, Haifa, Israel, 1986.

AUTHOR

Julian is Associate Professor at Massey University, New Zealand. Her core research focuses are; cybersecurity (identity management, intrusion detection, trustworthy system, cloud storage, applied cryptography) and privacy protection techniques (data anonymization and Homomorphic encryption) for big data analytics. Prior to Massey, she worked at CSIRO, the premiere Australian government research agency, for 15 years. She is an active member of several database, cyber security and health data informatics research communities and has published more than 50 articles in the leading conferences and journal venues including IEEE and ACM. She obtained a BBus(Comp) from University of Western Sydney, Masters and PhD from University of Sydney, Australia.