A Secure Communication Model for Expressive Access Control Using CP-ABE

Jayam Modi, Manav Prajapati, Abhinav Sharma, Ravi Ojha, and Devesh Jinwala
(Corresponding author: Devesh Jinwala)

Computer Engineering Department, S V National Institute of Technology, India
Computer Engineering Department, S V National Institute of Technology, India
Ichchhanath, SURAT-395 007, Gujarat, India
(Email: dcjinwala@gmail.com)
(Received June 13, 2015; revised and accepted Jan. 11 & Mar. 3, 2016)

Abstract

Attribute Based Encryption is a technique that associates user's attributes with keys. Data is encrypted using a specific policy and only those keys whose attributes satisfy that policy are allowed to decrypt it. In this paper, we propose a secure communication model based on Ciphertext Policy Attribute Based Encryption (CP-ABE). This model allows Role Based Access Control for documents without the use of a secure server to enforce the access policies. We propose a scalable implementation for key revocation and user attribute updation with improved flexibility. Our method uses a key revoke-list and key-version to achieve this. We show the implementation using the CP-ABE toolkit, an open source library that implements the CP-ABE scheme. We also show how confidentiality, integrity and source authentication is achieved in our model.

Keywords: Access rights, CP-ABE, expressive access control, secure communication model

1 Introduction

Information has been a valuable resource ever since humans began to communicate and like all other resources it needs to be protected. With the advent of the Internet and computing technology, digital means for exchanging information gained importance. Millions of people connected to Internet exchange information of potentially crucial nature. The methods used to secure this transfer of information have evolved over the years.

In this era of Internet, it is inevitable for various service providers like Google and Facebook to store sensitive personal information of users on servers. Considering the variety and importance of this information, there is a risk of an attack on these servers. This leads to concerns about compromise of personal data. To avoid such a compromise, the data can be stored in an encrypted form on the servers. This ensures that the privacy of the data remains intact. The task of selective sharing of information and access control now becomes a big challenge. Traditionally, a trusted server used to be employed in order to enforce access control but the data must be stored in unencrypted form on such a server. Public Key Infrastructure can be used to enforce access control over encrypted data by creating a trust model as discussed in [10]. In a PKI based model, when a user wants to selectively share the data, he must encrypt it with the public keys of each and every intended recipient. This is not a feasible option in many scenarios. When data is to be shared with a large group of users or the intended target audience determined by some attributes is not fully known, the PKI based model cannot be used. Another problem with this model is that the users higher in the access hierarchy have to store a large number of keys. This problem was solved to some extent using the method proposed in [1].

In [17], the idea of Attribute Based Encryption was introduced. Several schemes were proposed to achieve fine grained access control [9]. These scheme overcame the limitations faced by the model proposed in [10]. In an ABE based model, the data can be stored in an encrypted form on a server. This breakthrough lead to further developments in Role Based Access Control(RBAC) using ABE. Consequently, CP-ABE and KP-ABE encryption schemes were developed. A survey of these ABE schemes and access structures and their comparisons in cloud environment has been given in [12]. When the attributes are at different levels, the CP-HABE, which is a hierarchical CP-ABE scheme proposed in [14] proves to be useful.

1.1 Our Contribution

In this paper, we propose a secure communication model that can be used for selective sharing in an unsecure storage server environment. We achieve this by using CP-

ABE. Our model provides a scalable method for revoking keys and updating attributes of keys. We use a similar key revocation technique as in [6]. We attempt to address some issues in key revocation and dynamic attribute updation. Our scalable key revocation mechanism allows to effectively revoke a user's key immediately without much overhead by using revoke list. But the revocation process is completed lazily for a large batch of revoked keys. Our proposed mechanism provides complete freedom of choosing when to perform this lazy completion. It can be done when a threshold number of revocations have accumulated or any arbitrarily set time period has elapsed. We enable key attribute updation using the revocation mechanism itself. In our scheme, all key attributes can be assumed to be dynamic in nature. Also, our scheme does not require changing the public and master key pair of CP-ABE. We discuss the working of the proposed model in Section 3. We also provide pseudocodes in Section 4 to show clearly how our model can be implemented. We show how our model ensures confidentiality, integrity and authentication under some attack scenarios in Section 5.

2 Theoretical Background and Related Work

2.1 Theoretical Background

Cryptography is used to secure the communication between two parties. The earliest form of cryptography was secret key cryptography, which involved the use of a secret key that was known to both the parties before the exchange of data. As the size of networks and organizations grew bigger, the quadratic growth in the number of keys required for secure communication lead to serious concerns. Thus, when the idea of public key encryption was proposed by Diffie and Helman in [7], it was promptly accepted and as a result many different public key encryption schemes were developed.

A few emerging applications like cloud services often demand that the access to data should be governed by a policy wherein only specific individuals are granted access to the data. In such cases, there is a need for a cryptographic scheme that allows only those users whose attributes satisfies a decryption policy to decrypt the data. In Public Key cryptography, there is a single Private Key that can decrypt the data encrypted by the corresponding Public Key. Attribute Based Encryption [17] was introduced as an attempt to overcome this limitation.

Shamir in [18] defined a technique known as Identity Based Encryption that enabled any pair of users to communicate securely and to verify each other's signatures without exchanging private or public keys, without keeping directories and without using the services of a third party. In this scheme, the public key of the receiver is a combination of the receiver's attributes and it is computed by the sender with the help of publicly known attributes of the receiver. This eliminated the need for key exchange and therefore prevented man-in-the-middle attacks as opposed to public-key schemes. The private key of a user is generated by the key generation center after proper identity check of the user.

Attribute-based encryption (ABE) was an approach proposed by Sahai and Waters in [17]. In traditional public-key cryptography, a message is encrypted for a specific receiver using the receiver's public-key. But in large organizations, often there is requirement of a technique that allows members to specify access policies for restricting data to groups of intended recipients. This can be achieved by using a trusted server to store data. The server can check certification of a user before granting him access to files. A major drawback of this method is the security of the server. ABE aims to achieve secure selective sharing while removing the dependency on servers with access control mechanisms. The access control logic is embedded in the encryption technique and thus encrypted data remains confidential even if the storage server is untrusted.

There are two types of Attribute Based Encryption, namely Key-Policy Attribute Based Encryption (KP-ABE) and Ciphertext-Policy Attribute Based Encryption (CP-ABE). [9] provides a scheme to implement KP-ABE. In KP-ABE, ciphertexts are associated with sets of descriptive attributes, and users' keys are associated with policies. In key-policy ABE, the encryptor exerts no control over who has access to the data it encrypts, except by it's choice of descriptive attributes for the data. Rather, it must trust that the key-issuer issues the appropriate keys to grant or deny access to the appropriate users.

CP-ABE was first presented in [3]. In CP-ABE, a user's private key is associated with a set of attributes and the access policy is specified in the ciphertext. A user can decrypt an encrypted text if and only if his attributes satisfy the policy specified in the ciphertext. The policy can be built using conjunctions, disjunctions and (k, n) threshold gates. The private keys can be obtained by a user even after the data has been encrypted. Thus the actual set of users that can decrypt a ciphertext is not needed to be known at the time of encryption. This allows the incorporation of future users who may obtain a key that will satisfy the policy of the encrypted text and hence be able to read the data.

All the above efforts are shown to be special cases of Functional Encryption [4]. The term Functional Encryption was first seen in [11], disguised in the form of predicate encryption. Functional encryption is a scheme which allows a user to gain knowledge about a specific function of the encrypted text. The data is encrypted with a public key pk . A master secret key is held by a trusted authority. It can generate secret key sk_f corresponding to function f. The user having sk_f can compute the func-

tion f on any encryption of x. There are four phases in a functional encryption system - setup, keygen, encryption and decryption. The setup phase generates a public key and the master secret key. Keygen phase generates the secret key sk_k. Encryption phase encrypts a message x with the public key. Decryption phase enables user to compute $F(k, x)$ from the encrypted message. Functional encryption systems have a wide range of applications today like spam filtering on encrypted mail, expressive access control and mining on large datasets.

2.2 Related Work

The issue of user key revocation in CP-ABE is not addressed often in the proposed schemes. Majority of the authors who propose schemes focus on proving the security of their scheme. Piretti, Traynor and McDaniel in [15] roughly addressed the issue of attribute revocation for the first time. They suggested that each attribute should be valid only within a particular time-frame. After the validity of the attribute expires, the system administrator will release latest version of the attribute. The user updates his key based on the latest available version of the attribute. To revoke an attribute, the latest version of the attribute will not be released. The major problem with this solution is that of time synchronization between the system administrator and the user. To overcome this shortcoming, Bethencourt in [3] proposed that every key of a user should have an expiration date. A user will be able to decrypt the message only if the date of encryption of message is less than the expiration date of the user's key.

In [20], it is proposed that whenever an attribute needs to be revoked, the key generation authority will redefine the master key components of the revoked attributes. The corresponding public key components are also redefined. The user's secret keys need to be updated for data access. The new data is encrypted using the new public key. To perform these updates, proxy re-key's [13] are generated by the authority. Using these re-key's, the proxy servers can update the existing ciphertexts on the storage sever as well as the user's secret keys. This maintains backward compatibility in the system. This method transfers the load of the authorities onto the proxy servers leading to better performance than the methods proposed in [3] and [15].

Chen and Gerla in [5] proposed a fading function based method for implementing dynamic attributes. In their method, the concept of a fading function, F(x,y) was introduced. This function takes two parameters as input, the attribute name and the time at which its value is to be determined. It then outputs a unique value based on these two parameters. If the sender sends a message at time $t1$, the receiver will be able to decrypt that message at time $t2$ if and only if F(attribute, $t1$) = F(attribute, $t2$).

Weber in [19] proposed a method for incorporating the type of dynamic attributes whose values can be expressed in a list. He proposed that each attribute should be converted into a group element and then those elements should be translated into appropriate components of the private key. These components are transferred to the users device and stored in a secured compartment of the device that cannot be accessed by the user. All these attributes are bound together using a common random factor during the key generation process resulting in blinding of each key component. This in turn also blinds the ciphertext when the attributes are used in it. During the decryption process, the same common random factor is used to unblind the ciphertext. The malicious users cannot combine the components of the different keys in the same manner as an authentic user and thus the decryption algorithm will fail.

Chuha, Roy and Stoev in [6] use the concept of negative attributes to allow immediate key revocation, and the revocation process is completed lazily after a fixed time slice expires. To handle key attribute update, they propose that two separate access tress should be used for encryption process. One tree is for static attributes and other is for dynamic attributes. The tree of dynamic attributes is connected via a dummy node in the main access tree. After encryption using the main tree, the part of ciphertext that corresponds to dynamic attributes is separated out and re-encrypted using the access tree for dynamic attributes. The receiver applies original key to remaining part of ciphertext and obtains a key from the local key server for decrypting the dynamic attributes part of the cipher-text. Our model uses a similar technique but allows more flexibility and efficiency for key revocation by providing a mechanism to complete the revocation process after an arbitrary time limit or upon reaching a threshold number of revocations. Also, our scheme uses a simple technique for key attribute updation using the revocation mechanism. This allows making all key attributes dynamic.

Doshi in [8] proposed that for updating an attribute, the user should return his old secret key and the CA will give the user the new secret key to the user after verifying the new value of the attribute. The keygen algorithm takes old values from old secret key. An algorithm for using this technique in semi-trusted environment is also discussed in it.

3 Proposed Secure Communication Model

We propose a Secure Communication Model that allows expressive access control without the use of a secure storage server. The model has been developed using the cpabe-toolkit [2].

The CP-ABE toolkit provides four command line tools to perform the various operations of the CP-ABE scheme proposed in [3]. They can be used manually or can be invoked by larger systems. The four command line tools are:

- cpabe-setup - generates a public key and a master secret key;
- cpabe-keygen - generates a private key with a given set of attributes;
- cpabe-enc - encrypts a file according to a policy, which is an expression in terms of attributes;
- cpabe-dec - decrypts a file using a private key.

Our proposed model provides the following functionalities:

- Sending file to intended audience - A user can specify the attributes of the intended audience while sending a file. The model ensures that only the intended audience will be able to view the file.
- Receive files - A user can receive the files intended for him.
- Revoke access rights of some user - This is necessary if some user is no longer a part of the network and should not have access to the network's files.
- Update access rights of some user - This is necessary if the role of some user in the network changes.

Figure 1 shows a use case diagram of the model.

3.1 Components of the Proposed Model

The model has three entities, namely the Repository, Key Generation Center (KGC) and the users. The Repository and the KGC interact with the users to perform several tasks.

The Repository is a central server accessible to all. It is assumed that this server is not secure. The repository stores the encrypted files sent by all users, along with the timestamp when each file was uploaded at server, sending user's id and the minimum `Key_Version` required to decrypt the file. The Repository has a Public-Private key pair. All data sent from repository is signed using its Public key.

The Key Generation Center (KGC) performs the tasks related to key management. It stores the user's attributes and performs tasks such as initial key distribution, key revocation, distributing `Revoke_List` and `Active_Key_Version`, key renewal and updating user's attributes. The KGC has a Public-Private key pair. All data sent from KGC is signed using its Public key.

Each user has access to KGC and Repository. They have their own private key (i.e. cp-abe secret key), KGC and Repository's public key and the CP-ABE public key with them.

A private key in CP-ABE is associated with a set of attributes. In our proposed model, each private key has two types of attributes - User Attributes and Essential Attributes. User Attributes describe the user. E.g. department, name, experience, salary, etc. Essential Attributes are used to implement the model features. They are key_id and `Key_Version`. For a key to decrypt a file, in addition to satisfying the constraints on user attributes, it also needs to satisfy the constraints on essential attributes.

Each user has a `user_id` which uniquely identifies him within the organization. Each key has a `key_id` that uniquely identifies a key. At any given point of time, each `user_id` may be associated with only one `key_id`. Each `key_id` is uniquely associated with a fixed set of attributes. So, the `key_id` associated with a user has to be changed if the attributes of the user are changed.

At any point of time, the whole system will have an `Active_Key_Version`. It is a positive integer that is used for implementing the key revocation feature of the model. It starts from 1 and can only be incremented. Every key also has a `Key_Version` as one of its attributes which may be less than or equal to the `Active_Key_Version`.

3.2 The Proposed Model

When a new user enters the organization, he is authenticated at the KGC. His attributes are stored in a database at the KGC and a private key is provided to him. The public key of KGC and Repository is also provided to him. It is assumed that these functions are done through direct physical contact. A user must maintain the secrecy of his private key.

Further updates in the private key don't require direct physical contact as the KGC can simply encrypt the updated private key using cp-abe with policy such that only the concerned user may be able to decrypt it.

3.2.1 Send a File

For encrypting a file, the following data is required:

1) cp-abe Public key - It is publicly available and each user has a local copy on their own machine.
2) `Revoke_List` - This is the list of `key_ids` which have been revoked since the last `Active_Key_Version` update.
3) `Active_Key_Version` - This is the `Active_Key_Version` as discussed in Section 3.1.

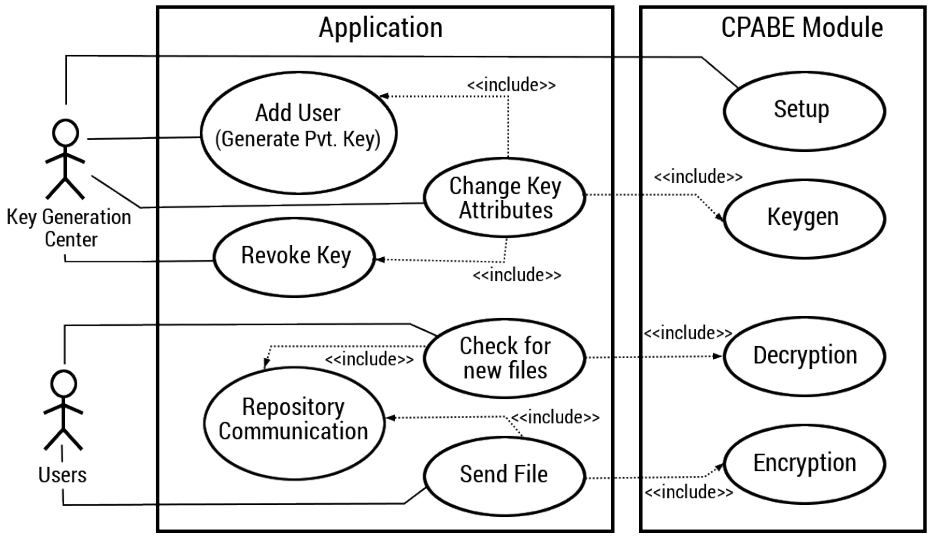

Figure 1 Use case diagram for the communication model

The `Revoke_List` and `Active_Key_Version` are stored at the KGC and are fetched every time a file needs to be encrypted.

The following steps are followed when a user uploads a file to repository:

1) User obtains the current `Revoke_List`, and `Active_Key_Version` from the KGC.
2) User select the policy file. The policy file contains a boolean formula that describes the User Attributes of the intended audience.
3) User generates the augmented policy file. In this step, the boolean formula in the policy file is augmented with constraints on the essential attributes. It includes the following:
   - minimum `Key_Version` required to decrypt the file, which is the `Active_Key_Version`. E.g. If the `Active_Key_Version` is 2, then version >= 2 is used;
   - list of revoked `key_ids`. The `key_id` that tries to decrypt the file should not be any of these. E.g. If the `Revoke_List` is (4, 6, 10), then (`key_id` != 4 and `key_id` != 6 and `key_id` != 10) is used.

   Thus, the (augmented policy) = (original policy) and (constraints on essential attributes). E.g. (augmented policy) = (original policy) and ((`key_id` != 4 and `key_id` != 6 and `key_id` != 10) and version >= 2).
4) User encrypts the selected file using cp-abe with policy as the augmented policy generated.
5) User sends 'Upload Request' to Repository along with his `user_id`.
6) Repository generates Asymmetric Key Pair (K1, K2) such that K1 and K2 are inverse of each other as done in RSA [16].
7) Repository encrypts K1 using cp-abe with policy such that only the concerned user's key be able to decrypt it.
8) Repository signs the encrypted data with his Public Key and sends it to the user.
9) User receives signed and encrypted K1 from Repository. He verifies the signature and decrypts K1.
10) User signs the encrypted file that he wants to upload with K1.
11) User uploads the signed and encrypted file along with the minimum `Key_Version` required to decrypt (which is the `Active_Key_Version` at the time of encryption) to the repository.
12) Repository receives the file, verifies the signature using K2 and stores it along with the upload timestamp and the minimum `Key_Version` required to decrypt the file.

It should be noted that a key with `Key_Version` less than the `Active_Key_Version` at the time of encryption of a file will not be able to decrypt the file. Such a key should be renewed before trying to use it to decrypt a file.

3.2.2 File Refresh

The user may periodically check the repository for new files intended for him. The following steps are performed in this operation:

1) User sends the File Refresh Request to the repository. In the request, the user also sends the timestamp of the last file refresh done by him.
2) Repository selects all files that were uploaded after the given timestamp along with the upload timestamp and the minimum `Key_Version` requirement of each file. Repository signs this data with his Public Key and sends the signed data to user.
3) User receives data from Repository and verifies the signature.
4) If the maximum of `Key_Version` requirement of all received files is greater than the `Key_Version` of the key possessed by the user, then he requests for an updated key from the KGC otherwise the next step is skipped.
5) (skipped if not required) The KGC generates another private key using cp-abe module for the user using his attributes (that were stored at KGC) and the `Active_Key_Version` of system. KGC encrypts the new key using cp-abe with policy such that only the concerned user be able to decrypt it. KGC signs the encrypted key with its Public Key and sends it to user. If the `key_id` of the user has been revoked, the KGC doesn't return any new key.
6) (skipped if `key_id` was revoked) User receives the signed and encrypted updated key from KGC. He verifies the signature and decrypts it with his old key.
7) User tries to decrypt each file using cp-abe module one by one using his private key (which may or may not be updated in the above step).
8) User deletes the files that couldn't be decrypted and can view those that were successfully decrypted.

3.2.3 Key Revocation

The KGC receives request from the administration for revoking a certain key. The following steps are performed at the KGC:

- KGC adds the `key_id` to the `Revoke_List`. Now whenever, a user requests the `Revoke_List`, this new list will be sent. So, when the user encrypts the file, the augmented policy will make sure that the none of the revoked keys can decrypt the file.
- It marks the `key_id` in its database.

This is a temporary fix for revoking keys as it is not scalable. The size of the `Revoke_List` will keep on increasing and lead to increased overhead. When a certain number of revoke keys have accumulated OR a fixed time period has passed, the following process will be done by KGC:

1) KGC increments the `Active_Key_Version` of the system.
2) KGC sets the `Revoke_List` to empty.

Now, whenever a file is encrypted, the new `Active_Key_Version` will be used to construct the augmented policy. As the existing keys have old `Key_Version`, they will not be able to decrypt it. The users may then ask the KGC for key renewal. They will be given their new keys, which will have the same attributes as their old key, but with the `Key_Version` incremented. The revoked users marked in the database will not be issued new keys.

Note that the user can still use his revoked key to decrypt only those messages that had been encrypted before his key was revoked.

3.2.4 Update Attributes

When the KGC receives request to update attributes of a particular user, the following steps are performed:

1) KGC finds the current `key_id` associated with the concerned user and revokes that key.
2) KGC generates a key with a new `key_id`, new attributes and `Active_Key_Version`.
3) KGC encrypts the updated key using cp-abe with policy such that only the old key of concerned user be able to decrypt it.
4) KGC signs the encrypted updated key. KGC sends the data to user.
5) User receives the data, verifies the signature and decrypts the updated key using his old key.

Note that even after receiving his new updated key, the user still possesses his old key. This old key can be used to decrypt only those messages that had been encrypted before his attributes were updated.

4 Pseudocodes

This model has been implemented using socket programming. There are three modules - User, Repository and KGC. The KGC and Repository modules run on a server and service requests sent by User module. The user can invoke commands for sending files or doing a file refresh through his modules. Calls are made to the CP-ABE toolkit to perform various functions.

The list of functions, invoked in various pseudocodes, along with their description is as follows:

- **cpabe-keygen(masterkey, public_key, attributes)** - CP-ABE module function that returns a private key associated with given attributes.

- **cpabe-enc(public_key, plain_text, policy)** - CP-ABE module function that returns encrypted file with given policy.
- **cpabe-dec(public_key, private_key, encrypted_text)** - CP-ABE module function that returns decrypted file if the provided key satisfies the policy.
- **request(request_type, receiver, ...)** - sends the specified request to the receiver with optional arguments and returns response from receiver.
- **response(data)** - sends data in response to the current request.
- **send(data, receiver)** - sends the data to specified receiver.
- **receive(data)** - receives data from current connection.
- **verify(attributes)** - verify if the passed attributes are correct.
- **new_keyID()** - generates a new unique key_id.

The following notations are obeyed in the pseudocodes:

- $\mathbf{(msg)_{k1}}$ - msg encrypted using Public Key cryptography with key k1. Denotes encryption, if k1 is public key. Denotes signing, if k1 is private key.
- $\mathbf{\{msg\}_{pol}}$ - msg encrypting using cpabe-enc with policy pol.

4.1 Repository Module

1) **Response to File Refresh Request** - This procedure in Pseudocode 1 is invoked while receiving new files.

Pseudocode 1 Response to Refresh Request

Input:
last_refresh_TS : Timestamp of last file refresh done by user

procedure PROCESS_REQUEST('Refresh Messages', last_refresh_TS)
  pkt ← ' '
  for all msg whose TS is > last_refresh_TS **do**
    pkt ← pkt + (msg, sender_user_id, upload_TS, key_version_required)
  end for
  signed_pkt ← $(\text{pkt})_{\text{REPO\_priv\_key}}$
  Response (signed_pkt)
end procedure

2) **Response to Upload Request** - This procedure in Pseudocode 2 is invoked when the user sends File Upload Request.

Pseudocode 2 Response to Upload Request

Input:
user_id : user_id of the user who sent Upload Request

procedure PROCESS_REQUEST('Upload Request', user_id)
  (K1, K2) ← generate Asymmetric key pair
  sender_key_id ← key_id_map[user_id]
  policy ← 'key_id=sender_key_id'
  $\{\text{K1}\}_{\text{policy}}$ ← cpabe-enc (pub_key, K1, policy)
  signed_msg1 ← $(\{\text{K1}\}_{\text{policy}})_{\text{REPO\_priv\_key}}$
  Response (signed_msg1)
  Receive (signed_msg2)
  $\{\text{msg}\}_{\text{policy}}$ ← $(\text{signed\_msg2})_{\text{K2}}$
  store ($\{\text{msg}\}_{\text{policy}}$, Current TS, Key_version, user_id)
end procedure

4.2 User Module

1) **Essential Constraints Generation** - This procedure in Pseudocode 3 is invoked by send file method.

Pseudocode 3 Essential constraints generation

Input:
Revoke_List : List of key_id that are revoked
AKV : Active_Key_Version of the system

procedure ESSENTIAL_CONSTRAINTS_GEN (Revoke_List, AKV):
  essential_constraints ← ' '
  for all x in Revoke_List: **do**
    essential_constraints ← essential_constraints + 'and key_id ! =x'
  end for
  essential_constraints ← essential_constraints + 'and key_version >= AKV'
  return essential_constraints
end procedure

2) **Send File** - This procedure in Pseudocode 4 is invoked when the user decides to upload file to Repository.

3) **File Refresh** - This procedure in Pseudocode 5 is invoked when the user wants to receive new files.

Pseudocode 4 Send file

Input:

msg : File that is to be upload to Repository

policy : Boolean formula denoting which users the file is intended for

1: **procedure** SEND_FILE(msg, policy):
2: signed_pkt1 ← Request ('Revoke_List and Active_Key_Version' , KGC)
3: (Revoke_List, AKV) ← $(\text{signed\_pkt1})_{\text{KGC\_pub\_key}}$
4: essential_constraints ← Essential_constraints_gen (Revoke_List, AKV)
5: augmented_policy ← '(' + policy + ')' + essential_constraints
6: $\{\text{msg}\}_{\text{augmented\_policy}}$ ← cpabe-enc (pub_key, msg, augmented_policy)
7: signed_pkt2 ← Request ('Upload Request', REPO, user_id)
8: $\{\text{K1}\}_{\texttt{key\_id=sender\_id}}$ ← $(\text{signed-pkt2})_{\text{REPO\_pub\_key}}$
9: K1 ← cpabe-dec (pub_key, priv_key, $\{\text{K1}\}_{\texttt{key\_id=sender\_id}}$)
10: signed_msg ← $(\{\text{msg}\}_{\text{augmented\_policy}})_{\text{K1}}$
11: send (signed_msg, REPO)
12: **end procedure**

4.3 KGC Module

1) **Response to Revoke_List and Active_Key_Version request** - This procedure in Pseudocode 6 is invoked when the user sends Request for `Revoke_List` and `Active_Key_Version`.

2) **Response to Attribute Updation** - This procedure in Pseudocode 7 is invoked when KGC has to update attributes of the user.

3) **Response to Update `key_version` Request** - This procedure in Pseudocode 8 is invoked when the user sends Update `key_version` request.

4) **Response to revoke user key request** - This procedure in Pseudocode 9 is invoked when a user's key is to be revoked.

Pseudocode 5 File Refresh

Input:

CKV : Current_Key_Version of key possessed by the invoking user.

last_refresh_TS : Timestamp of last File Refresh done by the user.

1: **procedure** REFRESH(CKV, last_refresh_TS):
2: signed_pkt ← Request ('Refresh Messages', REPO, last_refresh_TS)
3: Message_list ← $(\text{signed\_pkt})_{\text{REPO\_pub\_key}}$
4: Required_version← Max (version requirement of all files)
5: **if** Required_version > CKV **then**
6: signed_pkt ← Request ('Update Key_Version', KGC, user_id)
7: $\{\text{new\_key}\}_{\texttt{key\_id = requester\_key\_id}}$ ← $(\text{signed\_pkt})_{\text{KGC\_pub\_key}}$
8: new_key ← cpabe-dec (pub_key, priv_key, $\{\text{new\_key}\}_{\texttt{key\_id=requester\_key\_id}}$)
9: **end if**
10: **for all** enc_msg in Message_list **do**
11: (msg, status) ← cpabe-dec (pub_key, priv_key, enc_msg)
12: **if** status = fail **then**
13: delete msg
14: **else**
15: show msg
16: **end if**
17: **end for**
18: last_refresh_TS ← Max (TS of all files)
19: **end procedure**

Pseudocode 6 Response to Revoke_List and AKV request

1: **procedure** PROCESS_REQUEST(' Revoke_List and Active_Key_Version '):
2: msg ← List of revoked user + AKV
3: signed_msg ← $(\text{msg})_{\text{KGC\_priv\_key}}$
4: Response (signed_msg)
5: **end procedure**

5 Security Analysis of the Model

The KGC, Repository and user exchange information between them over an unsecure network to achieve the functionalities described in Section 3.2. We justify that our model ensures Confidentiality, Integrity and Authentication of the information exchanged.

Any message that the KGC or Repository send to a user is signed by their private key. Signing ensures integrity of information as well as authentication of its source. This signing process takes place in two cases-

- When the user requests `Revoke_List` and `Active_Key_Version` from KGC, the KGC signs the packet containing this data before sending it to the user.
- When the user requests Repository for File Refresh,

Pseudocode 7 Response to attribute updation

Input:

user_id : user_id of User whose attributes have to be updated

new_attributes : Updated attributes of the user

1: **procedure** PROCESS_REQUEST('Attribute updation', user_id, new_attribute)
2: verify(new_attributes)
3: attributes[user_id] ← new_attributes
4: old_key_id ← key_id_map[user_id]
5: Request('Revoke Key', KGC, old_key_id)
6: key_id_map[user_id] ← new_keyID()
7: new_key ← cpabe-keygen (master_key, pub_key, attributes[user_id] + key_id_map[user_id] + AKV)
8: policy ← 'key_id = old_key_id'
9: $\{new\_key\}_{key\_id\ =\ old\_key\_id}$ ← cpabe-enc (pub_key, new_key, policy)
10: signed-pkt ← $(\{new\_key\}_{key\_id\ =\ old\_key\_id})_{KGC\_priv\_key}$
11: Response(signed-pkt)
12: **end procedure**

Pseudocode 8 Response to update key_version request

Input:

user_id : user_id of user who sends the Update Key_Version Request

AKV : Active_Key_Version of the system

1: **procedure** PROCESS_REQUEST ('Update Key_Version', user_id):
2: requester_key_id ← key_id_map[user_id]
3: new_key ← cpabe-keygen (master_key, pub_key, attributes[user_id] + requester_key_id + AKV)
4: policy ← 'key_id = requester_key_id'
5: $\{new\_key\}_{policy}$ ← cpabe-enc (pub_key, new_key, policy)
6: signed_pkt ← $(\{new\_key\}_{policy})_{KGC\_priv\_key}$
7: Response(signed-pkt)
8: **end procedure**

the Repository signs all the files before sending them to the user.

When confidential data is sent to any user, encryption is done using cpabe with appropriate policy. This ensures confidentiality of information. This takes place in two scenarios.

- When the KGC sends a renewed key to user, the KGC encrypts the renewed key using cpabe-enc. The policy specified is such that only the intended user's old key can decrypt it.
- When a user has to send files to other users, the user encrypts data using cpabe-enc before uploading it to Repository.

Pseudocode 9 Response to revoke user key request

Input:

key_id : key_id of key that has to be revoked

Revoke_List : List of key_ids that have been revoked

Threshold : Maximum allowed size of Revoke_List

1: **procedure** PROCESS_REQUEST('Revoke Key', key_id):
2: Revoke_List ← Revoke_List + key_id
3: **if** Size (Revoke_List) > Threshold **then**
4: AKV ← AKV + 1
5: Revoke_List ← ϕ
6: **end if**
7: **end procedure**

When a user wants to send any data to the Repository, an asymmetric key pair is generated by KGC when user wants to upload a file. Integrity of file is achieved because the file is signed by user before sending to Repository. Authentication is achieved because only the user who sent the 'Upload Request' will be able to decrypt K1 which is to be used to sign the data. Confidentiality of file is ensured because the file is encrypted using cp-abe before signing. This approach avoids the need for public-private key pair for each user.

5.1 Attack Scenarios

Figure 2 describes the messages that are exchanged between various entities. We provide an analysis of how our communication model remains secure in the case of attacks carried out by an adversary on each of these messages.

1) Adversary fabricates Message 1 and sends it to KGC - In this case, there is no issue because `Revoke_List` and `Active Key Version` are public information and our model doesn't require them to be held secretly.

2) Adversary intercepts Message 2, modifies it and sends the corrupted message instead - The attack will be curbed as Message 2 is signed by the KGC. If the adversary tries to carry out such an attack, the user will detect that the message has lost its integrity.

3) Adversary poses as a legitimate user U1 and tries to upload a harmful file to Repository (Message 3) - An authentication mechanism is used by the Repository to verify the uploader. The Repository sends a message containing a key using which the uploader

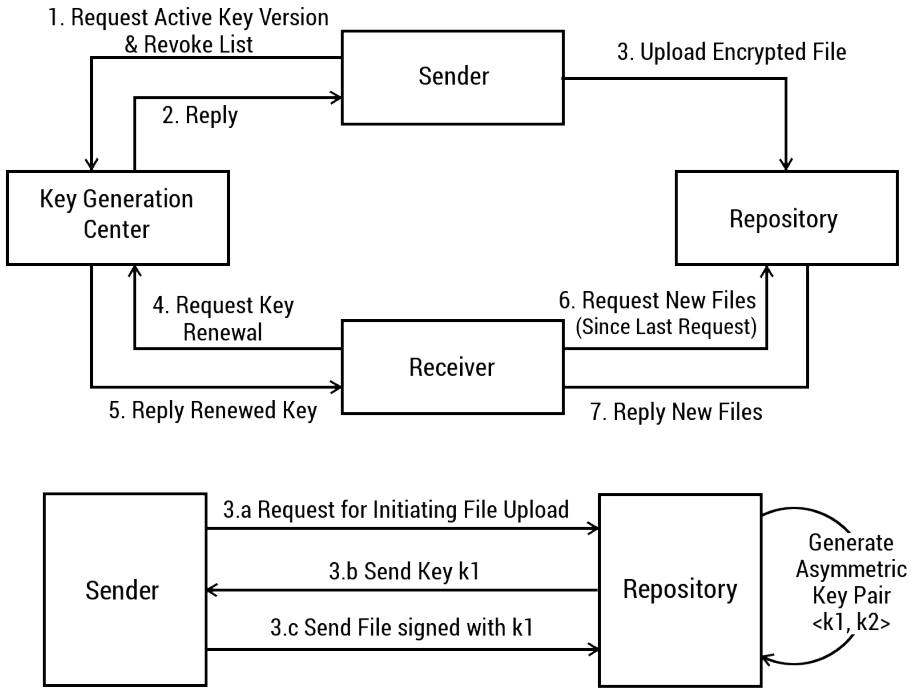

Figure 2 Communication between different entities

is supposed to sign his file. The Repository encrypts this message using cpabe-enc with policy such that only U1's key be able to decrypt it. Therefore, the adversary will not be able to extract the key from KGC's message and won't be able to sign the file.

4) A legitimate user perpetrates an insider attack by uploading a harmful file to Repository - Due to the authentication mechanism employed by the Repository, the uploader of each file is known. After detection of the harmful file, this information can be used to take action against the user.

5) Adversary fabricates Message 4 posing as a legitimate user U1 - The KGC encrypts the renewed key using cpabe-enc and a policy such that only U1's old key may be able to decrypt it. So, the adversary will not be able to decrypt the Message 5 from KGC. Therefore, the adversary will not be able to obtain U1's key. The only thing that this attack succeeds in doing is generating a futile response from KGC.

6) Adversary intercepts Message 5 from KGC to find out a user's key - Due to the same reason given in previous attack scenario, this attack won't work.

7) Adversary intercepts Message 7 that was being sent to user U1 OR Adversary sends Message 6 to Repository - The files uploaded on the Repository are encrypted using cpabe-enc. The Repository sends these encrypted files in response to a Message 6. The adversary will not be able to decrypt any files.

8) Adversary intercepts Message 7 that was being sent to user U1 and sends different files instead to U1 - The KGC signs the files before sending them to U1. Due to this, if an attacker tries to carry out such an attack U1 will detect that the files received by him are not from Repository.

In our proposed model, we rely on the security of the CP-ABE toolkit for the encryption and decryption process. We assume that data encrypted using the toolkit remains confidential. The toolkit implements the scheme proposed in [3].

6 Conclusion and Future Scope

The secure communication model proposed by us allows users to selectively share files among other users. It is more secure than using a server to enforce access control because in the event the Repository is compromised, our model ensures that the files would remain confidential. A user's key can be revoked, which effectively revokes all access rights of the user. The user's attributes can also be updated, which effectively changes his access rights.

As compared to a PKI-based approach, our model has the following advantages:

1) There is no need for managing multiple public keys using Certificate Authorities. There are only 3 public keys in our model: cpabe public key, KGC and Repository public key. These are available to all since the initial key distribution.

2) For sending file to N users only one encryption is required as opposed to N encryptions in case of PKI.

3) The sender can simply specify the attributes of the intended audience. As opposed to PKI, he doesn't need to know exactly who constitutes the intended audience. Due to this property of our model, there is no need for each User to store the list of all users along with their attributes.

These advantages hold as long as there exists a secure and scalable implementation of KGC.

The following are a few areas which can be worked upon to make our proposed model more secure and flexible:

1) Provide a mechanism to change the Master key-Public key pair of CP-ABE in case the Master key is compromised or a brute force attack is successful in discovering the Master key. It is a challenge to incorporate this functionality while still allowing operations on files encrypted before the Master key change.

2) Formalize the mechanism to change public-private key pair of KGC and Repository.

References

[1] S. G. Akl and P. D. Taylor, "Cryptographic solution to a problem of access control in a hierarchy," *ACM Transactions on Computer Systems*, vol. 1, no. 3, pp. 239–248, 1983.

[2] J. Bethencourt, A. Sahai, B. Waters, *Ciphertext-Policy Attribute-Based Encryption*, Mar. 24, 2011. (`http://acsc.cs.utexas.edu/cpabe/`)

[3] J. Bethencourt, A. Sahai, B. Waters, "Ciphertext-policy attribute-based encryption," in *IEEE Symposium on Security and Privacy (SP'07)*, pp. 321–334, 2007.

[4] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *Theory of Cryptography*, pp. 253–273, Springer, 2011.

[5] N. Chen and M. Gerla, "Dynamic attributes design in attribute based encryption," in *Annual Conference of ITA (ACITA)*, University of Maryland, MD, 2009.

[6] M. Chuah, S. Roy, and I. Stoev, "Secure descriptive message dissemination in dtns," in *Proceedings of the Second ACM International Workshop on Mobile Opportunistic Networking*, pp. 79–85, 2010.

[7] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[8] N. Doshi and D. Jinwala, "Updating attribute in cp-abe: A new approach.," *IACR Cryptology ePrint Archive*, vol. 2012, p. 496, 2012.

[9] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pp. 89–98, 2006.

[10] A. Herzberg, Y. Mass, J. Mihaeli, D. Naor, and Y. Ravid, "Access control meets public key infrastructure, or: Assigning roles to strangers," in *Proceedings of IEEE Symposium on Security and Privacy (S&P'00)*, pp. 2–14, 2000.

[11] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *Advances in Cryptology (EUROCRYPT'08)*, pp. 146–162, Springer, 2008.

[12] C. C. Lee, P. S. Chung, and M. S. Hwang, "A survey on attribute-based encryption schemes of access control in cloud environments," *International Journal of Network Security*, vol. 15, no. 4, pp. 231–240, 2013.

[13] X. Liang, Z. Cao, H. Lin, and J. Shao, "Attribute based proxy re-encryption with delegating capabilities," in *Proceedings of the 4th ACM International Symposium on Information, Computer, and Communications Security*, pp. 276–286, 2009.

[14] X. Liu, J. Ma, J. Xiong, and G. Liu, "Ciphertext-policy hierarchical attribute-based encryption for fine-grained access control of encryption data," *International Journal of Network Security*, vol. 16, no. 6, pp. 437–443, 2014.

[15] M. Pirretti, P. Traynor, P. McDaniel, and B. Waters, "Secure attribute-based systems," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pp. 99–112, 2006.

[16] R. L. Rivest, A. Shamir, and L. M. Adleman, *Cryptographic Communications System and Method*, US Patent 4,405,829, Sep. 20, 1983.

[17] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Advances in Cryptology (EUROCRYPT'05)*, pp. 457–473, Springer, 2005.

[18] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Advances in Cryptology*, pp. 47–53, Springer, 1985.

[19] S. G. Weber, "Securing first response coordination with dynamic attribute-based encryption," in *IEEE World Congress on Privacy, Security, Trust and the Management of e-Business (CONGRESS'09)*, pp. 58–69, 2009.

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

Jayam Modi, Manav Prajapati, Abhinav Sharma and Ravi Ojha obtained their B.Tech. in Computer Engineering in 2015 from Sardar Vallabhbhai National Institute of Technology, Surat, India. The work discussed here was a team effort towards the fulfilment of their B.Tech.

degree and was achieved under the guidance of Dr. Devesh Jinwala.

Devesh Jinwala is a Professor at Sardar Vallabhbhai National Institute of Technology, Surat, India. His major areas of interest are Information and Communications Security, Privacy and Cryptography, Security in Resource Constrained Devices, Software Requirements Specifications, Load Distribution and Failure Tolerance in Distributed Systems and Algorithms and Computational Complexity.