DSP Re-encryption Based Access Control Enforcement Management Mechanism in DaaS

Xiuxia Tian[1], Xiaoling Wang[2], and Aoying Zhou[2]
(Corresponding author: Xiuxia Tian)
School of Computer and Information Engineering, Shanghai University of Electric Power[1]
2103 Pingliang Road, Shanghai 200090,China
Shanghai Key Lab. of Trustworthy Computing, Software Engineering Institute[2]
East China Normal University, Shanghai 200062, China
(Email: xxtian@shiep.edu.cn)

(Received October 08, 2010; revised and accepted June 12, 2012)

Abstract

With the popular use of service-oriented technologies, Database as a Service(DaaS) paradigm is becoming a more practical and useful model for those enterprises who can't afford the expensive DBMS products. However, access control management by the database service provider(DSP) in this paradigm is challenged because the DSP may be untrusted for the delegated data contents. So it is important to design an access control mechanism which can couple with the delegated encrypted database to efficiently improve the usability of the system and help to prevent theft of sensitive and critical data.

In this paper, we present a novel approach to implement flexible access control enforcement management by designing a *DSP* re-encryption mechanism. Our approach not only can implement the selective authorization on the encrypted data, but also can relieve the client users from the complex key derivation procedure. The underlying idea of our approach is that the DSP uses different re-encryption keys for users of the system to implement flexible access control enforcement management under the DSP re-encryption mechanism. We demonstrate the efficiency and security of our flexible access control enforcement management, in the end we analyze and resolve the possible attacks and information disclosure.

Keywords: DaaS; DSP re-encryption mechanism; Access control enforcement;selective authorization

1 Introduction

With the cheap costs of communication, the convenience of network storage and network connectivity, and easier access such as the S3 provided by Amazon or AppEngine provided by Google, many small enterprises(data owners in our approach, simplified for *DO*s) resort to delegate their data storage and database management to one or more *DSP*s. Database as a Service, DaaS for short, caters to these requirements and allows enterprises to delegate their data management and data storage to *DSP*s so as to relieve them from excess costs of employing DBA professionals and associated hardware and software.

Access control, an important security mechanism in traditional DBMS, allows different users to have different access privileges. However, the DaaS paradigm, different from traditional client-server architecture in which the server is trusted and responsible for designing and enforcing the access control policy, is challenged because the DSP himself/herself is untrusted and may be one of the internal attackers. Therefore in DaaS paradigm if the trusted data owner is responsible for filtering unauthorized access to delegated database at *DSP* for each user in client, he/she will become the communication and performance bottleneck. Therefore it is necessary to enforce selective authorization on delegated database by DSP, at the same time to guarantee the confidentiality of the delegated sensitive data.

Different from the proposed approaches [14, 15, 16, 34], in this paper we present a novel solution which implements the flexible access control enforcement management at *DSP* by combining the *DSP* re-encryption mechanism with the access control policy of data owner. The concept of *DSP* re-encryption mechanism descends from the concept of primitive proxy re-encryption[5, 6, 22, 23] cryptography. Our approach not only efficiently implements the selective authorization enforcement on the delegated encrypted database by *DSP* and the dynamic policy updating, but also relieves users in client from the complex key derivation procedure. In our approach, the *DSP* re-encryption mechanism allows *DSP* to re-encrypt the delegated ciphertexts, which are the encrypted data tuples under the data owner's public key, into different re-ciphertexts

under the re-encryption key for each legitimate user. The re-ciphertexts are only decrypted by the user who is a legitimate user of data owner and has the private key corresponding to his/her re-encryption key.

Contributions. The main contributions of our paper are listed below.

1) DSP re-encryption based approach is introduced, in which a novel *DSP* re-encryption mechanism is introduced and used to implement the flexible access control enforcement management by *DSP* in DaaS paradigm.

2) In our approach, the *DSP* can implement the selective authorization enforcement management by using delegated access control authorization tables and the re-encryption module.

3) Our approach is efficient. The users in client need little computation knowledge to derive keys for authorized tuples, but only need to use his own private key to decrypt all the authorized encrypted tuples filtered and returned directly by *DSP*.

4) Our approach provides dynamic policy updating and managing. Whenever the granting or revoking operations take place, both the data owner and the *DSP* update the access control authorization tables by invoking the standard SQL updating statement.

Paper Organization. The rest of this paper is organized as follows. In Section 2 we first describe the introduced DSP re-encryption based system architecture, and then Section 3 introduces our *DSP* re-encryption mechanism and the first level encryption completed by data owner. We describe the concepts of general access control and necessary authorization information in Section 4. Section 5 comes the DSP re-encryption based approaches. Then we demonstrate the dynamic policy updating in DSP re-encryption based architecture in Section 6. In Section 7 we perform the experiment evaluation. Section 8 shows the security analysis of our proposed approach. We show the related work in Section 9 and finally conclude the paper in Section 10.

2 System Architecture

In this section, as shown in Figure 1 we first introduce the basic DSP re-encryption based system architecture. Then we describe the entities involved in the system architecture. From Figure 1 we can see there are three entities in the black dot line rectangle: Data Owner(***DO***), Database Service Provider(***DSP***), and Data Requester(***DR***) respectively. From Figure 1 we also see that the new introduced *DSP* re-encryption mechanism includes five components in the black real line rectangle: $\overrightarrow{E}$, *RE*, *REKG*, *PKG and* $\overrightarrow{D}$. We will introduce those components in Section 3 in detail.

Now we first describe the concepts of each entity and then the main tasks that each entity should complete in the new DSP re-encryption based system architecture.

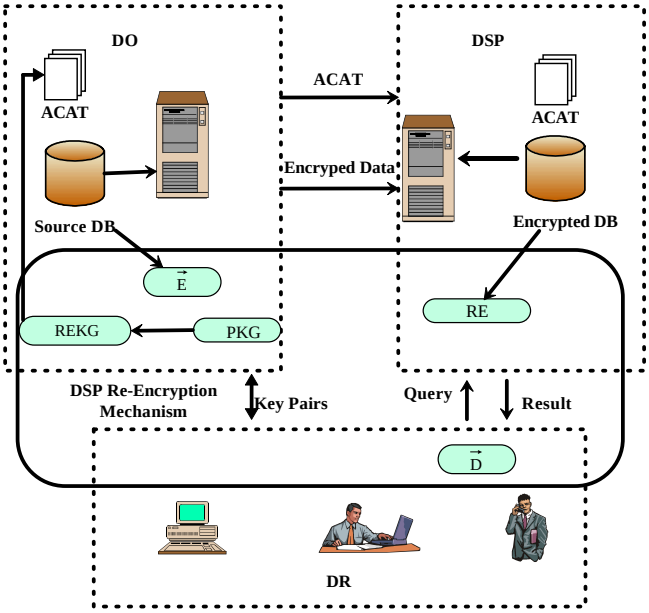

Figure 1: DSP Re-encryption Based System architecture

Data owner. Data owner(***DO***), may be an enterprise or individual who delegates his/her source database to a third party *DSP*. *DO* needs to complete the following four tasks.

- *Generating the private key.* One important task of *DO* is to generate the private key for each legitimate user of the system. The user is considered as the legitimate user if he/she can pass through the identity authentication of the system. In Figure 1, the component *PKG*, abbreviated from Private Key Generator, is used to implement the private key generation task. *PKG* is used to generate key pairs (the private key or secret key sk, the public key pk) for each legitimate user according to the identity information of the user. *PKG* is a polynomial time algorithm. In order to express clearly, we make the first capital character of user name as the parameter of sk and pk. For instance, (pk_A, sk_A) is the key pairs for user "Alice". The private key sk is forwarded to the corresponding user through a secure channel.

- *Generating re-encryption key.* Another task of *DO* is to generate an unique re-encryption key for each legitimate user. That is to say that each user only decrypts the authorized re-ciphertexts under his/her private key corresponding to his/her re-encryption key. This function is implemented by the component *REKG*, abbreviated from Re-Encryption Key Generator. *REKG* is also a polynomial time algorithm. The generated re-encryption keys are stored into an authorization table which is delegated to the *DSP* in a secure way.

- *Designing access control authorization tables*. *ACAT* or access control authorization tables. The third task of *DO* is to define the ACAT according to both the access control policy of *DO* and the outputs of *REKG*. There are at

least two access control authorization tables in Figure 5. The table "user-re-key" is used to store the re-encryption keys for each legitimate user and the table "user-tuple" is used to store which tuple a legitimate user can access to. For simplicity the access control policy is denoted as the access matrix in Figure 4. The access matrix can be created on a role-based or user-group based access control model, and it is based on the legitimate users in our approach. Both of the access control authorization tables need to be delegated to the *DSP* in a secure way.

- *Performing the first level encryption*. The last and important task of *DO* is to use a polynomial time algorithm E_i, $E_i \in \overrightarrow{E}$, to perform the first level encryption on tuples in Source DB. The first level encryption indicates that the source database(*Source DB* in Figure 1) can be transformed into an encrypted form(*Encrypted DB* in Figure 1) by using algorithm E_i. And the first level encryption is used to guarantee the confidentiality of delegated sensitive data against the *DSP* or other internal attackers. The encrypted tuples are augmented with additional information such as the encrypted tuple encryption-key. This allows the legitimate user to gain the tuple encryption-key easily, at the same time to protect the encryption-key against the *DSP*. This can be seen in $ekey$ column in table *Encrypted-Emp* in Figure 3(b), in which value for each tuple is the encrypted tuple encryption-key under the public key of *DO*.

Database service provider. The database service provider(*DSP*) is usually a professional database company and is responsible for the query response, access control enforcement and regular maintenance issues. *DSP* needs to complete the following two tasks.

- *Maintaining ACAT*. The first important task of *DSP* is to maintain the *ACAT* which includes at least two authorization tables in our approach. Both authorization tables can be updated flexibly and conveniently in term of the requirement of *DO*. The updating can be completed through the standard SQL statements. In order to avoid the disclosure of the $rekey$ during the transmitting we use the public key of *DSP* to encrypt the value of column "rekey" in table "user-re-key".

- *Performing the second level encryption*. The second most important task of *DSP* is to complete the second level encryption on the encrypted tuples in Encrypted DB. The second level encryption is implemented by the component *RE*, abbreviated for Re-Encryption, which is used to enforce the selective access control for different legitimate users. *DSP* re-encrypts the authorized encrypted tuples by using the re-encryption keys in authorization table "user-re-key" and the *RE* component. The result value of re-encryption on encrypted tuple is called as re-ciphertext. The re-ciphertext can only be decrypted by using the legitimate user's private key corresponding to his/her re-encryption key.

Data Requester. The Data Requester(***DR***) may be a PDA, PC, Mobile Phone, or any other electronic equipment. The *DR* needs to do the following two tasks.

- *Implementing the query transformation*. Query transformation function in client is used to transform the submitted query of user into a privacy preserving query form by using the correct public keys and additional index information, such as the bucket-id for bucket index. The component to implement this function is omitted in our architecture because our emphasis is on the access control enforcement mechanism.

- *Performing the decryption*. The second task of *DR* is to decrypt the re-ciphertexts from *DSP* and get the corresponding authorized plaintext tuples. Decryption is a polynomial algorithm D_i from the algorithm sets $\overrightarrow{D}$, $D_i \in \overrightarrow{D}$, and is used to decrypt the re-ciphertext under his/her private key.

3 The DSP Re-encryption Mechanism and the first Level Encryption

Figure 2 demonstrates the new introduced *DSP* re-encryption mechanism for flexible access control enforcement management and shows the information flows among the five components of *DSP* Re-Encryption mechanism. The *DSP* in our system is semi-trusted as that in the proposed approaches [14, 16]. The *DSP* can correctly maintain the delegated authorization tables, but maybe an internal attacker and could breach the confidentiality of delegated data.

3.1 *DSP* Re-encryption Mechanism

Definition 3.1 *A* DSP *re-encryption mechanism allows the* DSP *to re-encrypt a ciphertext c_m into different re-ciphertext rc_m. c_m is the encrypted value of any plaintext tuple m by using the data owner's public key pk_{DO} and a polynomial algorithm $E_i \in \overrightarrow{E}$. And rc_m is the encrypted value of c_m by using the RE module and the re-encryption key $rek_{DO \to user}$ of the legitimate user from the authorization table in ACAT.*

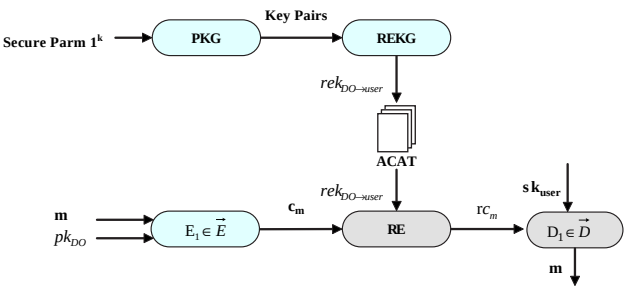

Figure 2: DSP re-encryption mechanism

As is shown in Figure 2, the *DSP* re-encryption mechanism is composed of five components: $\overrightarrow{E}$, *RE*, *REKG*, *PKG* and

$\overrightarrow{D}$. The *rekeys* as the input of RE module, generated by the *REKG* and stored into an authorization table of *ACAT*, are used to enforce the selective authorization. The selective authorization enforced re-ciphertext rc_m can only be decrypted by using the corresponding legitimate user's private key sk_{user}, such as private key sk_A for the legitimate user "Alice". The subscript arrow from *DO* to $user$ in $rek_{DO\rightarrow user}$ only allows the re-encryption from the DO to user, not from the user to DO. This is an one-way delegation.

For convenience, in the following we assume that m is any plaintext data such as the tuple in Source DB or encryption-key for one tuple. Suppose E_1 is the chosen standard first level encryption algorithm. By inputting pk_{DO} and m, $E_1 \in \overrightarrow{E}$ outputs a ciphertext c_m. The ciphertext is encrypted again by using the *RE* module in terms of the delegated ACAT to get the selective authorization enforced re-ciphertext rc_m. The rc_m can only be decrypted by the authorized user under his/her private key sk_{user} and the standard decryption algorithm D_1. Now we describe the function and usage of the components in Figure 2 respectively.

- $PKG, \overrightarrow{E}, \overrightarrow{D}$. They are the standard key pairs generation, encryption, and decryption algorithms respectively. $\overrightarrow{E}$ and $\overrightarrow{D}$ are the sets of standard encryption and decryption algorithms. The *PKG* algorithm outputs a key pairs (pk_{user}, sk_{user}) for the legitimate user by inputting the system security parameters 1^k. Generating the key pairs for user "Alice" can be expressed as:

  $$PKG(1^k) \rightarrow (pk_A, sk_A)$$

  The relationship between the first level encryption algorithm E_1 and the second level decryption algorithm D_1 for user "Alice" can be expressed as:

  $$E_1(pk_{DO}, m) \rightarrow c_m$$
  $$D_1(sk_A, rc_m) = m$$

  where rc_m is the encrypted value of c_m by using the re-encryption key and the RE module described in Section 5.

- *REKG*. *REKG* is an algorithm in the *DO* and is used to generate the corresponding re-encryption key. $rek_{DO\rightarrow user}$ is for the legitimate user by inputting the key pairs:

  $$(pk_{DO}, sk_{DO}, pk_{user}, sk_{user})$$

  or

  $$(pk_{DO}, sk_{DO}, pk_{user})$$

  Suppose the user is "Alice" and "A" is representative of "Alice", then the following is true:

  $$REKG(pk_{DO}, sk_{DO}, pk_A, sk_A) \rightarrow rek_{DO\rightarrow A}$$

- *RE*. The *RE* algorithm in *DSP* outputs the re-ciphertext rc_m on inputting the $rek_{DO\rightarrow user}$ and ciphertext c_m. The following is the access control enforcement of encrypted tuple c_m by *DSP* for the requesting user "Alice":

  $$RE(rek_{DO\rightarrow A}, c_m) \rightarrow rc_m$$

Example 3.1 *In order to describe how the introduced DSP re-encryption mechanism combines into our system architecture well, we give the following information flow and the related entities between each flow.*

$$DO : E_1(pk_{DO}, m) \rightarrow c_m : DSP$$
$$DSP : RE(rek_{DO\rightarrow A}, c_m) \rightarrow rc_m : Alice$$
$$Alice : D_1(sk_A, rc_m) = m$$

The information flow above can be explained simply as follows: 1) **The first formula**. DO encrypts the sensitive data m in Source DB into c_m under public key pk_{DO} of DO and delegates c_m to DSP. Through this the DO completes the first level encryption by using E_1. 2) **The second formula**. When Alice submits a query to DSP, DSP re-encrypts the c_m into rc_m under the re-encryption key $rek_{DO\rightarrow A}$ of Alice and returns rc_m to Alice. Through this the DSP completes the second level encryption, that is the re-encryption, by using RE and implements the selective authorization enforcement by using the delegated authorization tables. 3) **The third formula**. Alice decrypts rc_m into m under her private key sk_A.

3.2 First Level Encryption by DO

In DaaS paradigm the *DSP* is viewed as untrusted for the privacy of delegated sensitive data. So the *DO* should transform the sensitive data in Source DB into the corresponding private form(the encrypted form) against the privacy disclosure. We introduce the formal definition for the transformation(first level encryption) and give the corresponding tables in Figure 3.

Definition 3.2 *The first level encryption E_1, takes place at DO. The plaintext input of the first level encryption, encrypted into the corresponding ciphertext under the public key pk_{DO}, can be any data tuple ti in the source database R, or a randomly chosen encryption key $randki$ for each tuple.*

$$DO : E_1(pk_{DO}, ti) \rightarrow c_{ti} : DSP$$
$$DO : E_1(pk_{DO}, randki) \rightarrow c_{randki} : DSP$$

Example 3.2 *Table in Figure 3(a) is a relation table* Emp *in some personnel database. For simplicity we assume that there is only one table in Source DB of DO. Because the sensitive data in Table* Emp *may disclose the privacy of users,* DO *should transform those data to the private form. Table in Figure 3(b) is the corresponding encrypted relation table,* Encrypted-Emp *for Table* Emp. *From the two tables we can see that they have the same number of rows. Generally, for each relation r over scheme*

$$R(A_1, \ldots, A_n)$$

in Source DB in DO *there is a corresponding encrypted map relation r^k in Encrypted DB over the following scheme.*

$$R^k(tid, ekey, etuple)$$

userid	name	salary
1021	Tina	2458.3
1022	Krsa	6824.8
1023	Mary	5246.5
1024	Smile	3621.8
1025	Dings	5672.7
1026	Hers	3457.1
1027	Thde	3612.5

(a) Emp

tid	ekey	etuple
1	C_{randk1}	C_{t1}
2	C_{randk2}	C_{t2}
3	C_{randk3}	C_{t3}
4	C_{randk4}	C_{t4}
5	C_{randk5}	C_{t5}
6	C_{randk6}	C_{t6}
7	C_{randk6}	C_{t6}

(b) Encrypted-Emp

Figure 3: Table Emp and its corresponding encrypted Table Encrypted-Emp

	t1	t2	t3	t4	t5	t6	t7
Mike	1	0	1	0	1	1	0
Jack	0	1	1	1	0	1	1
Kate	1	0	0	0	1	0	0
Jone	0	0	0	1	0	0	0
Mary	0	0	1	0	0	1	1

Figure 4: Access matrix

Encryption key k is the public key pk_{DO} of *DO* or a data encryption key $randki$ chosen randomly by the *DO*. The attribute tid is an unique identifier for the encrypted tuple. The $ekey$ attribute only needs to be introduced in our multi-encryption key(MultiEK) approach. $etuple$ is the collum for encrypted tuple, whose value is generated through two different ways. One way is to use algorithm E_1 to generate the *etuple* value by inputting the corresponding source data tuple in relation r and the public key pk_{DO}. The other way is to use a standard symmetric algorithm to generate the *etuple* value by inputting the corresponding source data tuple in relation r, and the random data encryption key $randki$ in our One encryption key(OneEK) approach and MultiEK approach. In order to accelerate query efficiency the additional index can be generated through technologies proposed before such as the partition based approach in [20] or order preserving encryption for numeric data in [3].

4 Access Control and Authorization Tables

Access control is an important security mechanism in DBMS. Recently, [10, 14, 16] explored the selective encryption to implement access control in DaaS paradigm. Their approaches are inefficient. They adopt key distribution based on a user DAG in which each user must derive all the authorized keys with regard to a public catalog of tokens. At the same time, former work doesn't support the selective access to the encrypted database which is encrypted by one data encryption key. We present a different DSP re-encryption based approach which presents a flexible access control enforcement management mechanism by DSP for selective authorization access in DaaS.

This part first introduces the general access control model, and then describes the corresponding authorization tables used for the flexible access control enforcement management by *DSP*.

4.1 Access Matrix

Access matrix A is a conceptual model which specifies the rights that each subject(user or process) "S" possesses for each object(data tuple in our context) "O". This is defined by the server in the *Client*-server architecture where the server is believed as trusted. A should be defined by the data owner in DaaS scenario. For simplicity, we assume the database is read only. Therefore, the access matrix has the characteristics that there is a row for each S, a column for each O and a cell A[s,o]=1 if S can read O. If S can't read O, A[s,o]=0. Access control lists(ACL_t), are associated with a data tuple "T", indicates that which subjects can access data tuple T. Capabilities lists(CAP_s) denotes that which objects the user S can access to.

Taking the following as an example, assume there are five legitimate users in our system who want to access tuples of table in Figure 3(a), the corresponding access matrix for Mike, Jack, Kate, Jone and Mary is in Figure 4.

Until now, there are the following three approaches to enforce the selective access authorization represented in the access matrix of Figure 4 in DaaS paradigm.

- *One data encryption key for each tuple*. This approach requires that a user must remember many keys. This makes the key distribution difficult. For example, if two or more users have the same access rights, the *DO* must distribute all the same keys at least two times to each user.

- *One encryption key for all tuples*. This approach is adopted widely in the currently proposed approaches [3, 18]. The drawback of this approach is that each user can access the whole delegated database. Although the key management is very simple, the data owner may suffer a performance bottleneck in order to filter all the unauthorized data tuples for each user.

- *Selective encryption*. This is a recently proposed approach [10, 14, 16] which manages key assignments based on a user hierarchy. Nodes on the hierarchy are all possible sets of users and a partial relationship is defined according to the subset relation between them. This approach requires users to derive all the keys for the authorized tuples in terms of a public catalog of tokens which consist of keys and its corresponding public tokens.

Our approach is different from the three above, it takes advantage of the strengths of each while mitigating their drawbacks. In our approach the *DO* should delegate some additional authorization tables to the *DSP* without disclosing any

userid	name	rekey
100801	Mike	dxvsd
100802	Jack	csvr
100803	Kate	cvt56
100804	Jone	cg7hs
100805	Mary	drth7

(a) user-re-key

userid	tid
100801	1,3,5,6
100802	2,3,4,6,7
100803	1,5
100804	4
100805	3,6,7

(b) user-tuple

Figure 5: Authorization tables

private information of users. The detailed method is introduced in Section 5.

4.2 Authorization Tables

There are two authorization tables which need to be created, "user-re-key" and "user-tuple".

- *user-re-key(**userid**, name, rekey).* This table in Figure 5(a) stores the re-encryption key information for each legitimate user. The values for $rekey$ attribute can be used to re-encrypt the encrypted tuples with regard to the authorization tables by *DSP*. Each legitimate user of the system has a unique identifier(Attribute **userid**). A name attribute whose value can be the same and for each legitimate user there exists a unique re-encryption key(attribute **rekey**).

- *user-tuple(userid, tid).* This table in Figure 5(b) stores the tuples identifiers(attribute **tid**) that a user(attribute **userid**) can access to in terms of the access control policy of DO.

Both of the two tables should be delegated to the *DSP* in a secure way, such as transmitted under the public key of the *DSP* or through a special secure channel. In order to describe the enforcement of the access control by DSP we give the *user-re-key* table in Figure 5(a) in which we assume 100801 is the userid of Mike, 100802 for Jack and so on. We then give the *user-tuple* table in Figure 5(b) which lists all the tuples a user has the rights to access in a single row in terms of the access matrix in Figure 4. For example, the user of 100801 can access tuples t1, t3, t5, t6 with regard to the first row in *user-tuple*, but the practical stored *user-tuple* is one row for each tuple.

5 DSP Re-encryption based Approach

In this section we first present the concept of access control enforcement management in DaaS, then give our proposed approach.

5.1 Access Control Enforcement Management

Definition 5.1 *Access control enforcement management. The DSP can make full use of the delegated access control authorization tables in Figure 5 from ACAT and the re-encryption polynomial algorithm in RE to enforce the selective access to the authorized data tuples.*

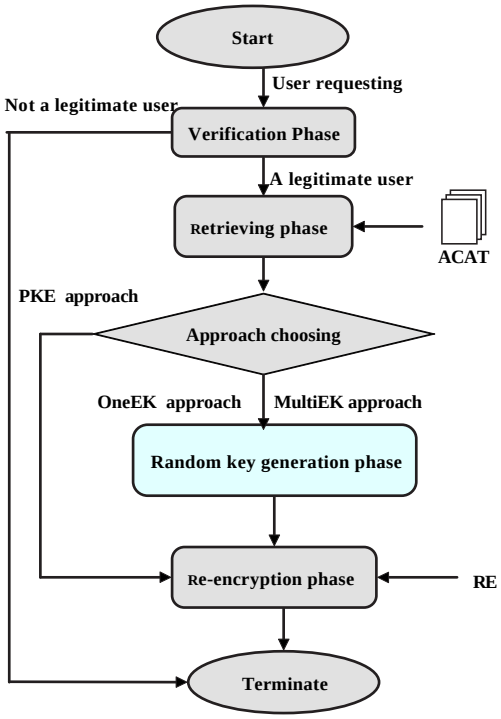

Figure 6: Access control enforcement process in DSP

Figure 6 is the access control enforcement management process in *DSP*.[1] The following is the concrete work in each phase.

- ***Verification phase***: Verify whether the requesting user is a legitimate user or not. If he/she is a legitimate user, then continue the process in the next step. However, if the verification fails the process movies directly to the *terminate* phase.

- ***Retrieving phase***: Fetch the unique value of *rekey* from table *user-re-key*, which resides in *ACAT*, into a variable *var-rekey* on the condition that the *userid* equals to the requesting user's id *req-userid*. The statement for fetching $rekey$ from table *user-re-key* is as follows:

SELECT rekey INTO var-rekey
FROM user-re-key
WHERE userid=req-userid

The values of attribute *tid* are then fetched into a record set *rectid*. This allows an authorized user to access many tuples in terms of his/her practical privileges. The statement for fetching tids of authorized tuples from table *user-tuple* is as follows:

SELECT tid INTO rectid
FROM user-tuple
WHERE userid=req-userid

[1]Our proposed DSP re-encryption based approaches follow this general work flow besides differences in some steps.

- ***Random key generation phase***: This phase only exists in the *OneEK approach* and the *MultiEK approach*. The main function of this phase is to generate a random key e' for the requesting legitimate user by using a Random Generator.
- ***Re-encryption phase***: According to the selected results *var-rekey* and *rectid* from the retrieving phase, for each tid in record set *rectid*, fetching the encrypted tuple c_{ti} from column $etuple$ in table $Encrypted - Emp$ in Figure 3(b), and then enforce c_{ti} as follows:

  $$RE(var - rekey, c_{ti}) \rightarrow rc_{ti}$$

  The operations in this phase are different in both the *OneEK approach* and *MultiEK approach*.

Example 5.1 *Mike and Jack both independently submit a query to retrieve some authorized data tuples in the delegated database. Even if they have the same access rights on tuple* $t3$*,* DSP *will do the following to enforce the selective authorization by using different* rekeys, $rek_{DO \rightarrow M}$ *for Mike and* $rek_{DO \rightarrow J}$ *for Jack*[2].

$$DSP : RE(rek_{DO \rightarrow M}, c_{t3}) \rightarrow rc^1_{t3} : Mike$$

$$DSP : RE(rek_{DO \rightarrow J}, c_{t3}) \rightarrow rc^2_{t3} : Jack$$

From the two computations above and the property of re-encryption algorithm we know that rc^1_{t3} is only decrypted by using the unique private key sk_M for Mike and rc^2_{t3} is only decrypted under the unique private key sk_J for Jack.

5.2 The PKE Approach

The *PKE* approach applies the concept of *DSP* re-encryption mechanism to the DaaS paradigm directly. Assume there is only one read only relation in Figure 3(a). *DO* encrypts each tuple in Figure 3(a) under his/her public key pk_{DO}. Figure 3(b) is the corresponding encrypted relation in which each value of attribute $etuple$ is the encrypted value for the corresponding tuple in Figure 3(a). The enforcement of selective authorization by *DSP* conforms to that in Figure 6.

Example 5.2 *From* user-tuple *in Figure 5(b), we know Kate is authorized to access tuples* $t1$ *and* $t5$*. From table* Encrypted-Emp *in Figure 3(b) we assume that* c_{t1} *and* c_{t5} *are the corresponding ciphertexts for tuples* $t1$ *and* $t5$ *under the public key of* DO*. The* DSP *does the following to enforce the selective authorization by using the re-encryption key* $rek_{DO \rightarrow K}$ *for Kate and the* RE *algorithm.*

$$DSP : RE(rek_{DO \rightarrow K}, c_{t1}) \rightarrow rc_{t1} : Kate$$

$$DSP : RE(rek_{DO \rightarrow K}, c_{t5}) \rightarrow rc_{t5} : Kate$$

After the second level encryption, that is the access control enforcement by using re-encryption, only Kate can decrypt rc_{t1} and rc_{t5} under her private key sk_{Kate} and the second level decryption algorithm D_1.

$$Kate : D_1(sk_{Kate}, rc_{t1}) \rightarrow t1$$

$$Kate : D_1(sk_{Kate}, rc_{t5}) \rightarrow t5$$

Even if the legitimate user Jone can obtain all the re-ciphertexts to Kate, he still can't get the original tuples from *DO*. This is because he hasn't the private key sk_{Kate} for Kate.

$$Jone : D_1(sk_{Jone}, rc_{t1}) \rightarrow t1' \neq t1$$

$$Jone : D_1(sk_{Jone}, rc_{t5}) \rightarrow t5' \neq t5$$

However, the *PKE* approach does have drawbacks. The speed of asymmetric encryption algorithm(RSA) is much more slower than that of the symmetric encryption algorithm(AES) when encrypting large mount of data [28]. [24] also conducted experiments on database and found that the performance and security of AES algorithm is better than that of RSA algorithm. Therefore, although this approach is simple and correctly enforces selective authorization by *DSP*, it isn't practical in the real database application. So in the following two sections we present our two improved approaches to implement the access control enforcement by DSP through introducing the symmetric encryption algorithm at the first encryption phase.

5.3 OneEK Approach

In the *OneEK* approach we introduce another pair of symmetric encryption(*SE*) and symmetric decryption(*SD*) algorithms, which are different from the asymmetric algorithms in $\overrightarrow{E}$ and $\overrightarrow{D}$. A symmetric algorithm needs one identical share key between the sender and the receiver to encrypt the plaintext and decrypt the ciphertext respectively. The *OneEK* approach is different from the *PKE* approach in two phases. One is the *Random key generation phase*, the other is the *Re-encryption phase*.

We demonstrate the *OneEK* approach from DO, *DSP* and *DR* in sequence as follows.

The *DO* first randomly chooses a symmetric encryption key e, and then he/she uses e to encrypt all tuples in *Emp* in Figure 3(a) and stores the corresponding encrypted values into $etuple$ column in *Encrypted-Emp* in Figure 3(b).

Example 5.3 *A tuple* ti *in Figure 3(a) is encrypted to the corresponding encrypted value* c_{ti} *in the* i^{th} *row of* $etuple$ *attribute in Figure 3(b) by using* SE *and* e*.*

$$DO : SE(e, ti) \rightarrow c_{ti} : DSP$$

And at the same time DO forwards the following value to DSP*:*

$$DO : E_1(pk_{DO}, e) \rightarrow c_e : DSP$$

The *DSP* stores c_e for the latter query and executes the following two different phases to implement the selective authorization enforcement for different legitimate users.

[2]Each legitimate user of the system has his/her unique re-encryption key, which leads to the same ciphertexts encrypted to different re-ciphertexts for different users even if they have the same access rights.

- *Random key generation phase*: Choosing another random key e' different from encryption key e to encrypt the authorized encrypted tuples. For example, encrypted tuple c_{ti} is changed into c'_{ti} by using the SE and e'.

$$DSP : SE(e', c_{ti}) \rightarrow c'_{ti} : user$$

- *Re-encryption phase*: Using the public key pk_{user} of the legitimate user and algorithm E_2 to encrypt e' to $c_{e'}$ so that the legitimate user can gain decryption key e' securely.

$$DSP : E_2(pk_{user}, e') \rightarrow c_{e'} : user$$

At the same time DSP uses re-encryption key $rek_{DO \rightarrow user}$ to re-encrypt c_e to rc_e as follows:

$$DSP : RE(rek_{DO \rightarrow user}, c_e) \rightarrow rc_e : user$$

DR firstly uses his/her private key sk_{user} and decryption algorithm D_2 to decrypt $c_{e'}$ and get e', then he/she uses e' and SD to decrypt c'_{ti} and get c_{ti}. He/she secondly gets the encryption key e through using his/her private key sk_{user} and decryption algorithm D_1. And he/she finally gets the real plaintext ti by using the symmetric encryption key e and decryption algorithm SD.

$$user : D_2(sk_{user}, c_{e'}) \rightarrow e', SD(e', c'_{ti}) \rightarrow c_{ti}$$

$$user : D_1(sk_{user}, rc_e) \rightarrow e, SD(e, c_{ti}) \rightarrow ti$$

The *OneEK* approach has the following main merits:

1) It makes full use of the different merits of symmetric encryption algorithm and the asymmetric encryption algorithm to improve the speed of data encryption and decryption. The efficient combination not only can enforce the selective authorization but can prevent against man in the middle attack which will be analyzed in detail in Section8.

2) It is easy to combine this approach into the existing approaches [18]. Lastly, the user in the *DR* needn't derive any keys in terms of a catalog of tokens like [15], only need remember his/her own private key sk_{user}.

However, this approach also has drawbacks. The largest drawback is that the disclosure of the one data encryption key may lead to the whole encrypted database exposed to both internal and external attackers. This is unacceptable, therefore we propose *MultiEK* approach to resolve this issue in the following section.

5.4 The MultiEK Approach

From Figure 6 we know that the MultiEK approach requires the same phases as those in the *OneEK* approach. However, the concrete operations in the *DO* and *DSP* have the following difference.

One difference is in the first level encryption where the *MultiEK* approach is to generate one random encryption key $randki$ for each tuple ti, not a random key *e* for all tuples. Although the encryption keys are as many as the number of tuples in Source DB, the key distribution isn't needed in our approach. We only need to add a new column $ekey$ to the corresponding encrypted table, such as the encrypted table *Encrypted-Emp* in Figure 3(b) for table *Emp* in Figure 3(a).

The *DO* randomly chooses an encryption key $randki$ for each tuple ti and encrypts tuple ti to the corresponding encrypted value c_{ti} in the i^{th} row of $etuple$ column in Figure 3(b) by using the symmetric encryption algorithm SE. Then DO encrypts $randki$ to the ciphertext c_{randki} in i^{th} row of $ekey$ column in Figure 3(b) by using asymmetric encryption algorithm E_1.

$$DO : SE(randki, ti) \rightarrow c_{ti} : DSP$$

$$DO : E_1(pk_{DO}, randki) \rightarrow c_{randki} : DSP$$

The *DSP* does the access control enforcement process as that in the *OneEK* approach, except for doing the following extra operation for the delegated encrypted c_{randki}.

$$DSP : RE(rek_{DO \rightarrow user}, c_{randki}) \rightarrow rc_{randki} : user$$

Our approach not only can take advantage of the *OneEK* approach, but also can avoid the *MultiEK* approach struggle with data security when an encryption key is compromised. It also has the advantage of not much more key distribution, even if all the encryption keys are different from each other. In conclusion, our system can work efficiently as that in the client-server environment in which the server can take over almost all computation. Using our new architecture allows the *DR* to be any light *DR*, such as PDA or Mobilephone.

6 Dynamic Policy Updating

There are three main kind of policy updating operations. These are composed of inserting or deleting a user, inserting or deleting a resource, and granting or revoking an authorization. Different from [16] and all the other proposed approaches where the delegated data tuples need to be decrypted under the old data encryption key, and encrypted and sent to the *DSP* again under the new data encryption key whenever the granting policy occurs. However, our approach only needs the *DO* to modify the authorization tables and to request the *DSP* to update the corresponding delegated authorization tables through invoking the standard SQL statement. This allows the access control policy of DO to be updated simultaneously, and therefore can largely reduce the requirement for high bandwidth and the costs for data communication. Our approach needn't the *DSP* do much operation to compute or derive the new key for the users or tuples when the policy updating. It only needs the *DO* to compute the re-encryption key for the new user once. The authorization tables in both the *DO* and the *DSP* are required to be maintained the same in our approach. The following shows the concrete changing when the three different policy updating operations takes place.

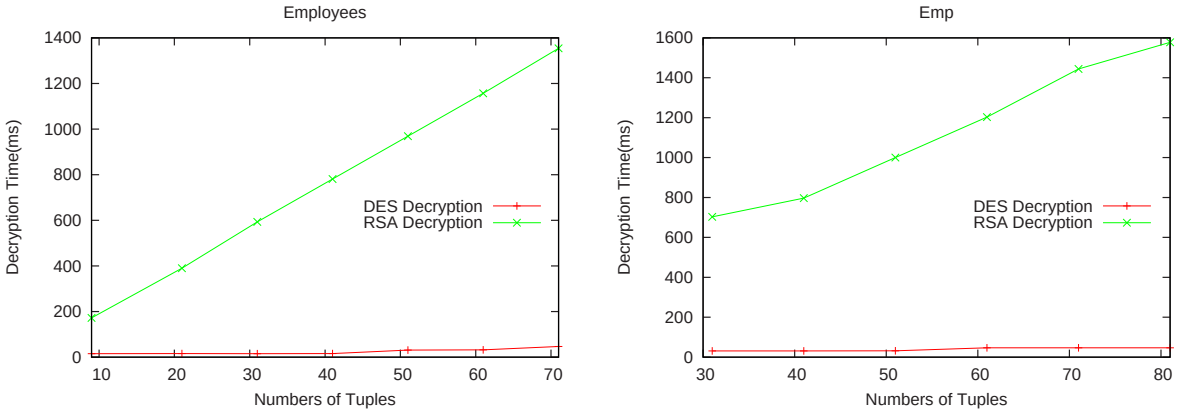

Figure 7: Two times DES Decryption and RSA Decryption of Employees and Emp

6.1 Inserting or Deleting A User

When inserting a new user u, the *DO* first generates the re-encryption key $rek_{DO \to u}$ for u by using the *REKG* algorithm. He/she then inserts a tuple about u and the corresponding $rek_{DO \to u}$ into the authorization table *user-re-key* through using the standard inserting SQL statement.

Example 6.1 *Assume the new user is Tutu, 100806, then* DO *does the following to generate the* rekey *for Tutu.*

$$REKG(pk_{DO}, sk_{DO}, pk_T, sk_T) \to rek_{DO \to T}$$

And then DO updates both the native and remote "user-re-key" table by using the following statement:

INSERT INTO user-re-key(userid,name,rekey)
VALUES(100806, "$Tutu$", $rek_{DO \to T}$)

When deleting a user u, the DO *only needs to ask the* DSP *to delete tuple for u from authorization table* user-re-key *and all tuples from authorization table* user-tuple *on the condition that the* userid *equals to the deleting* userid *of u.*

Example 6.2 *Assume the user is Mike, 100801, then the* DO *and the* DSP *do the following:*

DELETE FROM user-tuple
WHERE userid=100801

By executing this statement, all access rights accessed by user 10081 are deleted from table user-tuple*, and then do:*

DELETE FROM user-re-key
WHERE userid=100801

By executing this statement the user 100801 is deleted from table user-re-key*. After that the user 100801 can't access to the delegated database by DB any more.*

6.2 Inserting or Deleting A Resource

In our approach we assume the resources are tuples. However, they can be easily extended to the object resources, such as the tables or views. When inserting a tuple the *DO* can encrypt the tuple and send the encrypted tuple to the *DSP* without needing to update the policy, that is to say needn't to update the authorization tables.

When deleting a tuple, the *DO* only needs to request the *DSP* to delete all tuples in the authorization table *user-tuple* on the condition that the tid equals to the designated tuple tid, and then does the same for the authorization table *user-tuple* in his/her own.

Example 6.3 *Assume the tuple deleted is $t5$, then* DO *and* DSP *do the following:*

DELETE FROM user-tuple
WHERE tid=t5

6.3 Granting or Revoking An Authorization

Given the users and the tuples, any granting and revoking in our approach only require the *DO* to ask the *DSP* to insert or delete the tuple from the corresponding authorization tables. This is done from table *user-tuple* on the condition that the tid value in the tuple equals to the designated tuple tid value, and the *DO* does the same operation on the table *user-tuple* of his/her own.

7 Experiment Evaluation

The first goal of our experiment is to demonstrate the *PKE* approach, which applies the *DSP* RE-encryption mechanism into the DaaS paradigm directly. We intend to show this is not practical because the decryption of asymmetric algorithm(RSA) spends much more time than that of two times decryption of symmetric algorithm(DES). When compared to the approaches proposed by [14, 16], the *PKE* approach couldn't be chosen for the practical use. However, the subsequent two

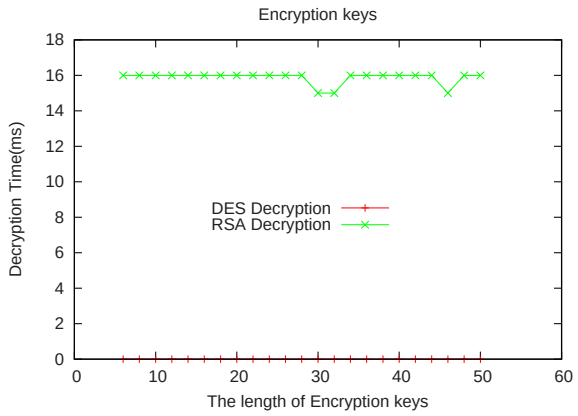

Figure 8: Different length of encryption string

improved approaches proposed have at least the same efficiency as the approaches proposed by [14, 16]. Also, our approach is better fit for the light user. This is because these methods require less computation for the *DR* in client.

In experiment, we use an Intel(R) Core(TM)2 Duo CPU, 2.33GHz PC with RAM 2G and 160GB hard disks as the server and the same configured PC as the *DR*. We use SQL server 2005 Express edition to store data on the server side and use the VS.NET 2008 as the integrated development environment, coding in $C\sharp$ with Framework $3 \cdot 5$. We choose a symmetric algorithm DES and an asymmetric algorithm RSA. There is no standard *DSP* re-encryption algorithm implemented, therefore we adopt RSA algorithm to imitate the *DSP* RE-encryption mechanism in our experiment evaluation, because they have the same work principle: public key for encryption and private key for decryption. However, they have an essential difference. The public and private key pairs of RSA belong to the same person, the public and private key pairs of *DSP* RE-encryption mechanism belong to different person, the public key for the delegator(*DO* in our approach) and the private key for the delegatee(the *DR* in our approach) are all different.

Data adopted. We use two classes of datasets to do our experiment. The first class of data is *Employees* in the well-known Northwind sample database typical used in SQL server. The second class of data is *Emp* which is a synthetic dataset generated by us. Although *Employees* and *Emp* may have many columns, we are only interested in the first three columns in each original tables and don't take care of the types of the attributes according to the columns. We formed a derived scheme *T(first, second, third)* which is then encrypted and stored in the corresponding encrypted table such as the *DESEmployeesEncTab* for table *Employees* and *DESEmpEncTab* for table *Emp*.

From Figure 7 we can see that the time of DES decryption is twenty times less than that of RSA decryption. Here, the DES decryption denotes that applying the DES algorithm two times by inputting different ciphertexts. The first time decryption is inputting the re-ciphertext which is corresponding to the second level DES encryption on the encrypted table of *Employees* or *Emp*, and then the second time decryption is by inputting the ciphertext which is corresponding to the first level DES encryption on table *Employees* or *Emp*. Using the asymmetric algorithm RSA to encrypt large data is more inefficient than that of symmetric algorithm DES, which couldn't be accepted by the online users. However the RSA encryption on string with length from 6 to 50 almost spends the same time such as 15 or 16 milliseconds demonstrated in Figure 8. This characteristic of RSA is typically used by us to encrypt the data encryption key in our two improved approaches and make our two approaches have at least the same efficiency as the proposed approaches by [16] and more efficiency in the *DR* than that of [14] because their approaches need much more encryption and decryption operations in order to protect the confidentiality of the policy.

8 Security Analysis and Possible Disclosure

This section first shows the security guarantee of our DSP re-encryption based approaches, then analyzes the possible information disclosure and the corresponding solution. In our context only the data owner can perform the first level data encryption operation as described in [33].

8.1 Security Analysis

Our three DSP re-encryption based approaches can provide the security guarantee as in [16] but have more flexible access control management. Almost the same access control management for one encryption key and multi-encryption keys. In our DSP re-encryption based approaches only the legitimate authorized user, who has the correct private key corresponding to the re-encryption key, can get the correct tuples. From Figure 1 we also know that *DO* and *DSP* have the same authorization tables which couldn't disclose more information than that in [15], because in theirs the *DSP* also needs to know all the access matrix information. Without this the *DSP* in theirs can't enforce the access control correctly and varies with the policy flexibly.

8.1.1 Privacy Guaranteeing

Even if the *DSP* knows the re-encryption keys for all users in the system, he/she couldn't derive the plaintexts from the delegated ciphertexts which only are decrypted by the very legitimate users. From Figure 2 we can see that only after the delegated ciphertexts are re-encrypted correctly by the *DSP* in terms of the appropriate re-encryption key, does the very legitimate user obtain the real plaintext tuples. The original plaintext tuples gain two level of protection from the data owner and the *DSP* respectively. Compared to the approaches in [15] our approaches do not require the user to derive a lot of keys in terms of a public catalog of tokens, but only need remember his/her own private key.

8.1.2 Man in the Middle Security

In proposed approaches [16], the same delegated tuples are always encrypted into the same re-ciphertexts due to the chosen unchanging keys by the *DSP*. In our two improved DSP re-encryption based approaches the *DSP* re-encrypts the delegated ciphertext tuples under a random chosen key only after he/she receives the request to access the authorized tuples. So the same ciphertext tuples will be re-encrypted into different re-ciphertexts from the previous, and thus can protect against the man in the middle attack to some extent. There is also the benefit that even if the same authorized users get different re-ciphertext tuples for the same ciphertext tuples because of the different re-encryption keys. This prevents a person from determining whether any two user have the same access rights in different time belong to the same user. The malicious user or the eavesdropper would be unable to get useful information through analysis the re-ciphertexts between the *DSP* and the legitimate users.

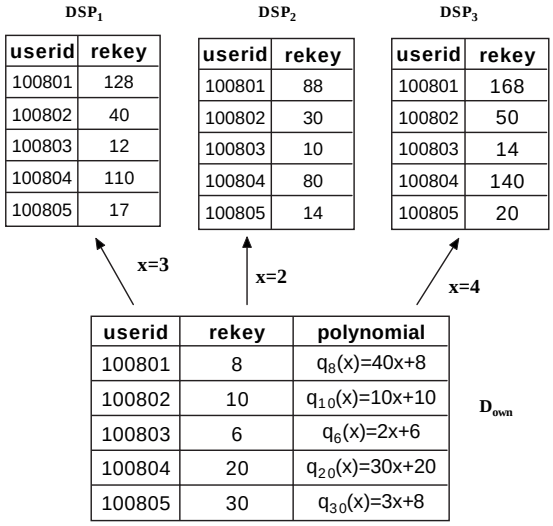

Figure 9: multi-DSPs based on the secret share scheme

8.2 Possible Disclosure

There must exist the possibility of the collusion between two entities such as one user and the *DSP* or the collusion of different users. If the collusion takes place in two different users, only the authorized tuples for these two users are disclosed. However, if the *DSP* colludes with a user, the collusion user may access to all the unauthorized tuples as long as the *DSP* re-encrypts each tuple by using the re-encryption key of the collusion user. Those disclosures are also true in [15]. Although they analyze the security evolution from different locked views, the different collusion users can access to the unauthorized tuples belonging to the other. The collusion between the user and the *DSP* will lead to all the tuples of the collusion user in terms of the public catalog of the tokens. The latter of theirs is better than ours, but they must distribute two keys for each user, one for the first level encryption and the other for the second encryption, and the access policy in the *DO* must be synchronized with the policy with the *DSP*. However, in our approach when the access control policy changes by using the revoking and granting operations, the DO only needs to update the corresponding authorization table in Figure 5(b), needn't to update the re-encryption key of the user. In order to avoid the information disclosure as much as possible we introduce two solutions to our architecture in the following.

From the above we know that the *DSP* should act correctly by re-encrypting all ciphertexts according to the delegated authorization tables and then the access control of the system can be flexibly enforced as that in the *DO*. This assumption is not always true, for example, *DSP* and multiple users become malicious. This can be correct to some extent by introducing multi-*DSP*.

- *Introducing a role-based* DSP *re-encryption mechanism*. A role-based *DSP* re-encryption mechanism[3] enables the *DO* to implement different access control policies for protecting the delegated ciphertexts against the malicious users of the system. The *DO* can classify the delegated data tuples in terms of the different roles required by practical situation and then delegate the decryption rights to the users who belong to the same role. Each *DSP* can work efficiently in the DSP re-encryption based architecture proposed by us. For example, there may be five roles in the system which require five $DSPs$ to perform the access control enforcement according to the re-encryption keys of the very role. In this situation *DO* can choose freely the trusted and good reputation *DSP* to manage his access control enforcement in terms of the roles. The next work in this way we will concentrate on how to choose appropriate number of $DSPs$ to realize the efficient access control management through using the role-based *DSP* re-encryption mechanism.

- *Introducing the secret share multi*-DSP. By using the secret share scheme [2, 28] among the multi-$DSPs$ to protect against the collusion between the user and the *DSP*. The basic motivation is that most of the $DSPs$ can honestly maintain the delegated secret share of a re-encryption key and the collusion couldn't happen so long as the number of collusion *DSP*s are less than the designated threshold k in a (k, n), $n > k$ threshold scheme.

  Suppose there is an example of $(2, 3)$ threshold scheme used to our $DSPs$, $k = 2$ and $n = 3$, Figure 9 demonstrates how to distribute the different secret shares among all the $DSPs$ of n in which we omit the unnecessary column such as the $name$ attribute for simplicity. Suppose that the five re-encryption keys are 8, 10, 6, 20, 30 respectively and the corresponding polynomials are $q_8(x) = 40x + 8$, $q_{10}(x) = 10x + 10$, the rest three referring to the Figure 9. So the share secret re-encryption keys stored in DSP_1, DSP_2 and DSP_3 are computed as follows. First assuming that the secret information cho-

[3]Here, each DSP can take over a role to be responsible for the delegated data having the same rights, which then reduces the danger of disclosing information among different roles.

sen by the data owner is 3 for DSP_1, 2 for DSP_2 and 4 for DSP_3. So the share secret of re-encryption 8 for the three $DSPs$ is

$$q_8(x) = 40x + 8 = q_8(3) = 40 \times 3 + 8 = 128$$
$$q_8(x) = 40x + 8 = q_8(2) = 40 \times 2 + 8 = 88$$
$$q_8(x) = 40x + 8 = q_8(4) = 40 \times 4 + 8 = 168$$

respectively, such as the first row of values of $rekey$ attribute for each *DSP* in Figure 9. The rest of the shares of the four re-encryption keys can be computed as above. In terms of the secret share scheme at least two $DSPs$ can combine to compute the corresponding re-encryption key through use of Lagrange interpolation and the secret information chosen by the data owner. Such as for x=3 and x=2, and know 40 from DSP_1 and 30 from DSP_2, through computing the Lagrange interpolation to get the re-encryption key 10. Using this method the data owner can reduce their dependency on some *DSP* and decrease the danger of the collusion between the user the *DSP*.

9 Related Work

In 2002 Hacigumus et al[18] first proposed the concept of DaaS and developed a prototype system NetDB2 which mainly resolved two important challenges about data privacy and performance in DaaS scenario. NetDB2 guarantees data privacy by adopting software or hardware encryption scheme on the delegated data and evaluated the system performance from three different granularity such as field, row and page. Although encryption can protect the privacy of the delegated data. In order to improve the usability of the delegated encrypted database most current proposed approaches[3, 19, 20, 21] are based on exploiting indexing information, which is stored together with the delegated encrypted database. This is to help the service provider select the returned data for the query without the need of decrypting the data. Especially the order preserving encryption function proposed in [3] can support range queries and is adopted widely in many subsequent schemes. Besides the privacy protection in the service provider ensuring the integrity and correctness of query results, the user is also needed in DaaS scenario. For example, the schemes in [26, 27, 30]. In order to allow an efficient query execution [1, 9] present solutions which exploit the combination of fragmentation and encryption to store data on a single server by minimizing the amount of encrypted data. However, because of the low speed of the software-based encryption [17, 28], [8] explore the hardware-based approach to support secure computation on both the *DR* and the service provider. [24] experiments database encryption efficiency through three different dimensions by adopting software encryption, hardware encryption, and hybird encryption respectively. Finally, they concluded that the hybird encryption on database-level is most efficient and can prevent theft of critical data and protect against threats such as the storage theft and storage attacks. Recently [31, 32] stated again the importance of a special secure hardware component-trust hardware and its application in most of the untrusted context.

Access control in Database is an important mechanism for implementing security and data privacy. There are many access control models to apply to different data management systems. [7] is a good paper which states the current challenge of database security and evolution of the access control model in different data management system. Due to the particularity of the DaaS scenario, there is no research about access control issue in DaaS until in 2007 Damiani et al [10] first addressed the problem of enforcing access control by exploiting selective encryption,that is to say, making use of different encryption keys for different data. In fact Damiani et al [13] in 2005 first presented an approach for the implementation of access control through selective encryption. Selective encryption is also adopted in the XML data publication [25] in which a framework for enforcing access control on published XML documents. This works by using different cryptographic keys over different portions of the XML tree. [12] makes use of the selective encryption to release information. At the same time with Damiani et al, Vimercati et al. [15, 16] proposed a novel two layer data encryption based on selective encryption to enforce the access control in a dynamic policy scenario. The inner layer is imposed by the data owner for the initial privacy protection against the unauthorized user and untrust provider and the outer layer is imposed by the service provider to reflect the policy modification against the unauthorized user. As a matter of fact in [15, 16] that data owner must at least semi-trust in the service provider or else the policy modification couldn't be enforced by the service provider. In order to efficiently distribute the keys to the least possible to authorized users [15, 16] adopt a key derivation method based on user hierarchy or directed acyclic graphs(DAGs). The authorized user needs to derive all the keys he is authorized in terms of a public catalog of tokens. Recently [14] proposed adding an encryption layer in the public catalog of tokens so as to avoid the information leakage of access control model of system. The scheme in [14] can be combined into the scheme [16] correctly and work more securely. [11] explores the management of metadata, the use of which can efficiently improve the usability of the system in DaaS scenario, but it requires the data owner to store much metadata for security. The recently proposed scheme in [35] does not require encryption of the same data(key) multiple times with the keys of different users or groups of users. It also significantly reduces the storage for storing the public parameters. In the paper proposed by [2] considered the database management as a service. [29] proposed how to guarantee the privacy of delegated policy and [4] presented how to combine different privacy approaches into outsourced data application. Although several proposed schemes above are capable of enforcing access control in DaaS scenario, almost all of them are subject to exploit by the same selective encryption to implement the access control based on the partial relationship of the users of the system.

10 Conclusions

With the computation capability improving greatly and the efficiency of scale economy, DaaS paradigm is adopted by var-

ious *DO*s. This includes medium or small enterprises, as well as individual users. However, there exists the potential security problems which must be resolved before its practical application. In this paper we address the problem of enforcing access control by *DSP* to make the system more usable by introducing new DSP re-encryption based approaches. Our approach efficiently combines a new *DSP* re-encryption mechanism with access control policy of *DO* in DaaS scenario. Moreover, the DSP re-encryption based architecture still satisfies the secure performance of the confidentiality and can reduce the computation complexity of the *DR*. At the same time our approach can eliminate the public catalog of tokens, but use some authorization tables. The *PKG* can be flexibly managed by a secure certificated authorization center or the *DO* himself/herself.

The subsequent research under this new architecture is on how to design the efficient query transformation in *DR*. The accessory mechanism such as the integrity and query guarantee by *DSP* and the efficient implementation of role-based and the secret share based *DSP* re-encryption mechanisms.

Acknowledgments

This work was supported by National Natural Science Foundation of China under grants(NO. 61202020, NO.61170085), Natural science foundation of Shanghai City(NO.12ZR1411900), Innovation Program of Shanghai Municipal Education Commission(NO. 12YZ147), Shanghai Education Key Curriculum Project(NO.20115308).

References

[1] G. Aggarwal, M. Bawa, P. Ganesan, and H. Garcia-Molina, "Two can keep a secret: a distributed architecture for secure database services," in *Proceedings of CIDR(CIDR 2005)*, pp. 186–199, Asilomar, CA, 2005.

[2] D. Agrawal, A. E. Abbabi, F. Emekci, and A. Metwally. "Datamanagement as a service:challenges and opportunities,". tech. rep., 2009.

[3] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proceedings of the ACM SIGMOD Conference(SIGMOD 2004)*, pp. 563–574, Paris, France, 2004.

[4] T. Allard, N. Anciaux, L. Bouganium, Y. L. Guo, L. Folgoc, B. Nguyen, P. Pucheral, I. Ray, and S. Yin, "Secure personal data servers: a vision paper," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 25–35, 2010.

[5] G. Ateniese, K. Benson, and S. Hohenberger, "Key-private proxy re-encryption," *Lecture Notes in Computer Science, Topics in Cryptology-CT-RSA*, vol. LNCS 5473, pp. 279–294, 2009.

[6] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," in *Proceedings of the Annual Network and Distributed System Security Symposium*, pp. 83–107, San Diego, California, 2005.

[7] E. Bertino and R. Sandhu, "Database security-concepts, approaches and challenges," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 1, pp. 2–19, 2005.

[8] L. Bouganim and P. Pucheral, "Chip-secured data access: confidential data on untrusted servers," in *Proceedings of the 28th VLDB Conference(VLDB 2002)*, pp. 131–142, Hong Kong, China, 2002.

[9] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Fragmentation and encryption to enforce privacy in data storage," in *Proceedings of ESORICS(ESORICS 2007)*, pp. 171–186, Dresden, Germany, 2007.

[10] E. Damiani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Selective data encryption in outsourced dynamic environments," *Electronic Notes in Theoretical Computer Science*, vol. 16, pp. 127–142, 2004.

[11] E. Damiani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Metadata management in outsourced encrypted databases," *Lecture Notes in Computer Science, Secure Data Management*, vol. LNCS3674, pp. 16–32, 2007.

[12] E. Damiani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, and P. Samarati. "Selective release of information in outsourced encrypted database,". tech. rep., 2005.

[13] E. Damiani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S.Paraboschi, and P.Samarati, "Key management for multi-user encrypted databases," in *Proceedings of the 2005 ACM workshop on Storage security and survivability*, pp. 74–83, New York, NY, 2005.

[14] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Pelosi, and P. Samarati, "Preserving confidentiality of security policies in data outsourcing," in *Proceedings of the 7th ACM workshop on Privacy in the electronic society*, pp. 75–84, Alexandria, VA, 2008.

[15] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "A data outsourcing architecture combining cryptography and access control," in *Proceedings of the 1st Computer Security Architecture Workshop*, pp. 63–69, Fairfax, VA, 2007.

[16] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Over-encryption: management of access control evolution on outsourced data," in *Proceedings of the 33rd VLDB Conference(VLDB 2007)*, pp. 123–134, Vienna, Austria, 2007.

[17] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu, "Privacy-preserving data publishing: A survey on recent developments," *ACM Computing Surveys*, vol. 42, no. 14, pp. 1–53, 2010.

[18] B. Iyer H. Hacigumus and S. Mehrotra, "Providing database as a service," in *Proceedings of 18th International Conference on Data Engineering(ICDE 2002)*, pp. 29–38, San Jose, California, 2002.

[19] H. Hacigumus, B. Iyer, and S. Mehrotra, "Ensuring integrity of encrypted databases in database as a service model," in *Proceedings of DBSec(DBSec 2003)*, pp. 61–74, Estes Park Colorado, CA, 2003.

[20] H. Hacigumus, B. Iyer, S. Mehrotra, and C. Li, "Executing sql over encrypted data in the database-service-provider model," in *Proceedings of the ACM SIGMOD(SIGMOD 2002)*, pp. 216–227, Madison, WI, 2002.

[21] B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries," in *Proceedings of the 30th VLDB Conference(VLDB 2004)*, pp. 720–731, Toronto, Canada, 2004.

[22] B. Libert and D. Vergnaud, "Tracing malicious proxies in proxy re-encryption," *Lecture Notes in Computer Science, Pairing-Based Cryptography-Pairing*, vol. LNCS 5209, pp. 332–353, 2008.

[23] T. Matsuo, "Proxy re-encryption systems for identity-based encryption," *Lecture Notes in Computer Science*, vol. LNCS 4575, pp. 247–267, 2007.

[24] U. Mattsson. "Database encryption-how to balance security with performance,". tech. rep., 2005.

[25] G. Miklau and D. Suciu, "Controlling access to published data using cryptography," in *Proceedings of the 29th VLDB conference(VLDB 2003)*, pp. 898–909, Berlin, Germany, 2003.

[26] E. Mykletun, M. Narasimha, and G. Tsudik, "Authentication and integrity in outsourced database," *ACM Transactions on Storage(TOS)*, vol. 2, no. 2, pp. 107–138, 2004.

[27] M. Narasimha and G. Tsudik, "Dsac: integrity for outsourced databases with signature aggregation and chaining," in *Proceedings of the 14th ACM ICIKM(ICIKM 2005)*, pp. 235–236, Bremen, Germany, 2005.

[28] B. Schneier, *Applied Cryptography(in Chinese)*. Beijing, China: China Mechine Press, 2006.

[29] N. Shang, M. Nabeel, F. Paci, and E.Bertino, "A privacy-preserving approach to policy-based content dissemination," in *Proceedings of 26th ICDE Conference(ICDE 2010)*, pp. 944–955, Long Beach, California, 2010.

[30] R. Sion, "Query execution assurance for outsourced databases," in *Proceedings of the 31st VLDB Conference(VLDB 2005)*, pp. 601–612, Trondheim, Norway, 2005.

[31] R. Sion, "Trusted hardware," in *Invited tutorial at the ACM Conference on Computer and Communications Security CCS*, Alexandria, VA, 2008.

[32] R. Sion and S. Smith, "Understanding and deploying trusted hardware," in *Invited tutorial at the USENIX Security Symposium*, San Jose, CA, 2008.

[33] Q. Tang, "Type-based proxy re-encryption and its construction," *Lecture Notes in Computer Science, Progress in Cryptology-INDOCRYPT*, vol. LNCS 5365, pp. 130–144, 2008.

[34] A. Velagapalli and M. Ramkumar, "Trustworthy tcb for dns servers," *International Journal of Network Security*, vol. 14, no. 4, pp. 187–205, 2012.

[35] A. Zych, M. Petkovi, and W. Jonker, "Efficient key management for cryptographically enforced access control," *Computer Standards and Interfaces*, vol. 30, no. 6, pp. 410–417, 2008.

Xiuxia Tian, Ph.D., Associate professor, Her research interests include database security, privacy preserving, applied cryptography.

Xiaoling Wang, Professor, supervisor of Ph. D.. Her research interests include technology of database management, web service.

Aoying Zhou, Professor, supervisor of Ph. D.. His research interests focus on data management and information system, inclusive of web data management, Chinese Web Infrastructure, web searching and mining, data streaming and mining, complex event processing and real time business intelligence, uncertain data management and applications, data intensive computing, distributed storage and computing, peer to peer computing and management, web service.