Missing a Trusted Reference Monitor: How to Enforce Confidential and Dynamic Access Policies?

Leila Karimi, Seyyed Ahmad Javadi, Mohammad Ali Hadavi(✉), and Rasool Jalili

Department of Computer Engineering,
Sharif University of Technology, Tehran, Iran
{l_karimi, ajavadi, mhadavi}@ce.sharif.edu,
jalili@sharif.edu

Abstract. Popularity of data outsourcing and its consequent access control issues such as dynamism and efficiency is the main motivation of this paper. Existing solutions suffer from the potential unlimited number of user keys, inefficient update of policies, and disclosure of data owner's access control policies. Using Chinese remainder theorem and proxy re-encryption together, in this paper, we propose an efficient access control enforcement mechanism based on selective encryption that addresses all the shortages. The overall architecture, required algorithms, and access control policy update are discussed. The mechanism is evaluated through simulation and, the given results are satisfactory.

Keywords: Access control · Selective encryption · Chinese remainder theorem · Proxy re-encryption

1 Introduction

Due to reducing communication costs and increasing volume of data, Database-As-a-Service (DAS) model, in which an organization outsources its data to a database service provider, becomes a popular paradigm. Although data outsourcing provides many benefits, it introduces new security concerns. The main concern is the storage of sensitive data on a site that is not under the direct control of the data owner. As a result, the data confidentiality and integrity can be compromised. Additionally, the enforcement of access control restrictions on the outsourced data is of main concerns; the issue on which we have concentrated in this paper. In fact, access control enforcement cannot be delegated to a server, which is not trusted enough to be aware of access policies and to enforce them. A promising solution is selective encryption, which couples authorization and encryption. It translates an access policy into an equivalent encryption policy so that only legitimate users are able to retrieve the decryption key of a protected resource.

In addition to maintaining a user hierarchy and a key derivation process, enforcing access control through selective encryption faces a couple of challenges. Firstly,

A. Movaghar et al. (Eds.): CNDS 2013, CCIS 428, pp. 92–104, 2014.
DOI: 10.1007/978-3-319-10903-9_8

updating access policies may cause high overhead to the data owner. It needs acquiring granted/revoked resources from the server, re-encryption of them, and resending them to the server. Secondly, privacy of access control policies may be violated due to the untrustworthiness of the remote server.

In this paper, we propose a solution to overcome the above shortcomings. In particular, we address the problem of key management using the Chinese remainder theorem, and of efficiently supporting policy changes using the proxy re-encryption scheme. Using these two schemes together also ensures the privacy of authorization policies, as it is an important requirement for the data owner.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 is dedicated to some preliminaries and basic concepts to our solution. Section 4 describes the proposed solution for efficient access control enforcement in detail. This section also demonstrates how changes in the access control policies can be handled efficiently. Section 5 describes our proposed prototype architecture. Section 6 analyzes our solution from the security and efficiency perspectives. Finally, Sect. 7 concludes the paper.

2 Related Work

Damiani et al. [1] proposed an access control mechanism for an outsourced database using selective encryption, where they reduced the number of user keys through key derivation. Vimercati et al. [2] proposed a two-layer data encryption to enforce access control for dynamic policies. In their scheme, a public token catalogue expresses key derivation relationships. However, the catalogue could leak information about the policies and data. To solve the problem, they proposed adding an encryption layer on the public catalogue of tokens [3]. There are several shortcomings related to their scheme. The algorithm of building key derivation structure imposes high computing overhead to the data owner. An update to the access control policy requires the users to obtain new keys derived from the rebuilt key derivation structure and also requires data re-encryption with the new keys. As the consequence, the scheme is not scalable. Tian et al. [4] introduce a new DSP re-encryption mechanism, which provides an efficient policy update and management approach. Ateniese et al. [5] proposed a secure distributed storage scheme based on proxy re-encryption. Their scheme depends on the existence of a semi-trusted server. Moreover, the system data security can be compromised due to the collusion of a malicious server and any single malicious user.

In 2005, Sahai et al. [6] proposed an access control scheme, referred to as threshold encryption, based on their introduced concept of Attribute Based Encryption (ABE). In the scheme, the owner encrypts his data and specifies an attribute set and a number *d*. Only a user with at least *d* attributes of the given attribute set can decrypt the retrieved data. Wang et al. [7] combined techniques of ABE, proxy re-encryption, and lazy re-encryption to delegate most of the computation tasks involved in the revocation of authorizations to an untrusted server without disclosing data content.

Chinese Remainder Theorem (CRT) was used to propose a general access control structure when the policy enforcement point is not trusted [8]. Tourani et al. [9] then

used above idea and proposed a CRT-based access control enforcement mechanism for data outsourcing scenario. Their solution allows updating policy changes, and access control policies are protected from being revealed to the server or the users.

3 Preliminaries

Let us review basic concepts and preliminaries, including CRT and proxy re-encryption before the introduction of our proposed access control enforcement mechanism.

3.1 Chinese Remainder Theorem (CRT)

Definition 1. *Chinese remainder theorem: for the system of simultaneous congruences in (1), in which* $k \geq 2$*, the positive integers* $n_1, n_2, \ldots, n_k$ *are pairwise relatively prime, and* $a_1, a_2 \ldots, a_k \in \mathbb{Z}$*, there exists a unique solution x, such that* $0 \leq x < n = n_1 n_2 \ldots n_k$.

$$\begin{cases} x \equiv a_1 \bmod n_1 \\ x \equiv a_2 \bmod n_2 \\ \quad \ldots \\ x \equiv a_k \bmod n_k \end{cases} \tag{1}$$

Kong et al. [8] used CRT and proposed a scheme to share a key with k users. Suppose K_r is the key we want to share it with users $u_1, u_2, \ldots, u_k$. K_{s_i} is the private key of user u_i. Then, x_{K_r}, the solution of simultaneous congruences in (2), can be used as a shared key of K_r.

$$\begin{cases} x_{K_r} \equiv E_{K_{u_1}}(K_r) \bmod n_{u_1} \\ x_{K_r} \equiv E_{K_{u_2}}(K_r) \bmod n_{u_2} \\ \quad \ldots \\ x_{K_r} \equiv E_{K_{u_k}}(K_r) \bmod n_{u_k} \end{cases} \tag{2}$$

When a user u_j is given a shared key x_{K_r}, he can compute K_r as follows:

$$E_{K_{u_j}}(K_r) = x_{K_r} \bmod n_{u_j} \tag{3}$$

$$K_r = D_{K_{u_j}^{-1}}(E_{K_{u_j}}(K_r)) \tag{4}$$

3.2 Proxy Re-Encryption

Definition 2. *Proxy re-encryption: Proxy re-encryption is a solution in which the data owner securely delegates the re-encryption mechanism to a proxy. The proxy re-encrypts the data without the need to decrypt any parts of the data.*

Syalim et al. [10] proposed a proxy re-encryption scheme for the symmetric ciphers, which is used in our mechanism to efficiently manage policy updates. Below, we introduce some primitive functions in symmetric proxy re-encryption scheme. Please refer to [10] for further details.

- *All or nothing transform (AONT)*: the algorithm converts a s-block message $M = m_0, \ldots, m_{s-1}$ to a pseudo message $M' = m'_0, \ldots, m'_{n-1}$ with n blocks, such that $n > s$ and any block of the original message cannot be retrieved if any block of the pseudo message is lost.
- *Perm*: the permutation algorithm *Perm* takes two sequences with the same size n as inputs and changes the order of the second sequence according to the first sequence, called "the permutation key." For example, the output of *Perm((3, 1, 2, 0),(a, b, c, d))* is *(d, b, c, a)*.
- *DePerm*: the algorithm takes two sequences as an input and changes the order of the second sequence to the form before being permuted by the *Perm* function using the first sequence as the permutation key. For example, the output of *DePerm((3, 1, 2, 0),(d, b, c, a))* is *(a, b, c, d)*.
- *FindCK*: the algorithm generates a conversion key *CK*, which can be used to convert a permuted sequence $POUT_A$ to another permuted sequence $POUT_B$. If $POUT_A$is the permuted form of *InSeq* with the permutation key KP_A and $POUT_B$ is the permuted form of *InSeq* with the permutation key KP_B, then the conversion key *CK* has the property: $Perm(KP_B, InSeq) = Perm(CK, Perm(KP_A, InSeq))$
- *PGen*: the permutation generator function *PGen* takes a key k and an integer n as inputs and generates a permutation key A with n elements. It is implemented using the encryption function *E(k, p)* that encrypts the plaintext p using the key k.
- *RandGen:* giving an integer n as an input, it generates an n-bit size random key.

4 Access Control Enforcement Mechanism

This section represents our proposed mechanism. At first, we describe a generic architecture for access control in the DAS model. Then, we describe our proposed mechanism based on the architecture.

4.1 Overall Framework

Figure 1 shows the system architecture for enforcing access control in the DAS model that involves following entities:

- Data owner: an organization that produces data and delegates its maintenance to the untrusted server.
- Server: an expertise database organization that receives encrypted data, is responsible to maintain it, responds to user queries, and helps the data owner in updating access control policies.
- User: an entity, whose queries are translated into queries over the encrypted data and sent to the server.

We assume that the server is honest-but-curious. It is honest in executing the protocols but is not trusted with the confidentiality of data and access control policies.

4.2 Initialization Phase

The following actions are performed by the data owner to outsource its data:

- For each user u_i of the system, the data owner generates a pair of keys $(K_{u_i}, K_{u_i}^{-1})$ and a modulus n_{u_i}.
- For each resource r of the system, the data owner generates a key K_r by which the resource is encrypted.

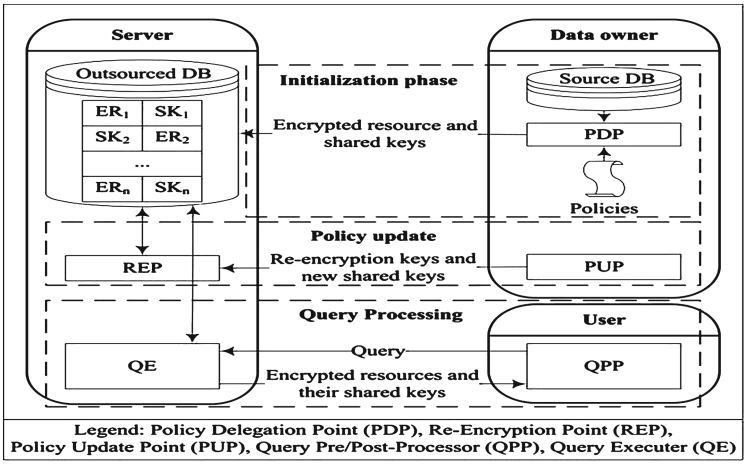

Fig. 1. Overall architecture of the proposed mechanism

- For each resource r of the system with access control list $\mathrm{ACL}(r) = \{u_1, u_2, \ldots, u_k\}$ and encryption key K_r, the data owner calculates the CRT solution x_{K_r} for the simultaneous congruences in (5).

$$\begin{cases} x_{K_r} \equiv E_{K_{u_1}}(K_r) \bmod n_{u_1} \\ x_{K_r} \equiv E_{K_{u_2}}(K_r) \bmod n_{u_2} \\ \quad \cdots \\ x_{K_r} \equiv E_{K_{u_k}}(K_r) \bmod n_{u_k} \end{cases} \tag{5}$$

For each resource r, the data owner calculates $c_r = \mathrm{SE}_{K_r}(r)$ and sends c_r and x_{K_r} to the server. Detailed explanation of the SE function is provided in Sect. 5.

4.3 Query Processing Phase

When a user u_i needs to retrieve a resource r from the server, he needs to perform the following steps with the server:

- User u_i generates a query and request r from the server.
- The server processes the query and sends c_r and x_{K_r} to u_i.
- If $u_i \in \text{ACL}(r)$ then he computes K_r as follows:

$$E_{K_{u_i}}(K_r) = x_{K_r} \bmod n_{u_i} \tag{6}$$

$$K_r = D_{K_{u_i}^{-1}}\left(E_{K_{u_i}}(K_r)\right) \tag{7}$$

- User u_i accesses r using its key: $r = \text{SD}_{\text{K}_\text{r}}(c_r)$. The function SD is further explained in Sect. 5.

If $u_i \notin \text{ACL}(r)$, he cannot retrieve the key K_r, so he cannot access r.

4.4 Policy Update

Policy update operations can be restricted to granting and revoking an authorization. Here we describe general steps of our mechanism for updating policies.

1) Grant: Granting a new user u_i an access to a resource r does not require the key of the resource to be changed but it needs the shared key x_{K_r} to be updated. Kong et al. [8] suggest an efficient way for this purpose. Consider simultaneous congruence equations in (8) with x_{K_r} as the solution and the simultaneous congruences in (9) which contain a congruence equation for the new user u_i with x'_{k_r} as the solution.

$$\begin{cases} x_{K_r} \equiv E_{K_{u_1}}(K_r) \bmod n_{u_1} \\ x_{K_r} \equiv E_{K_{u_2}}(K_r) \bmod n_{u_2} \\ \quad \cdots \\ x_{K_r} \equiv E_{K_{u_k}}(K_r) \bmod n_{u_k} \end{cases} \tag{8}$$

$$\begin{cases} x'_{k_r} \equiv E_{K_{u_1}}(K_r) \bmod n_{u_1} \\ x'_{k_r} \equiv E_{K_{u_2}}(K_r) \bmod n_{u_2} \\ \quad \cdots \\ x'_{k_r} \equiv E_{K_{u_k}}(K_r) \bmod n_{u_k} \\ \boldsymbol{x'_{k_r} \equiv E_{K_{u_i}}(K_r) \bmod n_{u_i}} \end{cases} \tag{9}$$

According to [8], the value of x'_{k_r} can be easily obtained by solving (10).

$$\begin{cases} x'_{k_r} \equiv x_{K_r} \bmod n_{u_1} n_{u_2} \ldots n_{u_k} \\ x'_{k_r} \equiv E_{K_{u_i}}(K_r) \bmod n_{u_i} \end{cases} \tag{10}$$

2) Revoke: Authorization revocation is more complicated than grant and requires changing the key of the resource. Consider a situation in which the data owner wants to revoke access to a resource r (encrypted with $\boldsymbol{K_r}$) from a user $\boldsymbol{u_i}$. The data owner and the server must go through following steps:

- The data owner updates ACL(r) by removing user u_i.
- The data owner generates re-encryption key REK_r using RKG function with K_r as an input.
- The data owner calculates the CRT solution x'_{k_r} according to the new ACL(r).
- The data owner sends REK_r and x'_{k_r} to the server and asks him to re-encrypt r.
- The server calculates c'_r using RE function with c_r and REK_r as inputs.

Preserving the confidentiality of r, this method re-encrypts r with a new key without imposing high computational overhead to the data owner.

5 The System Architecture

For implementing our proposed access control enforcement mechanism, we suggest the architecture shown in Fig. 2. According to Fig. 2, the proposed architecture consists of four main components discussed in the subsequent sections.

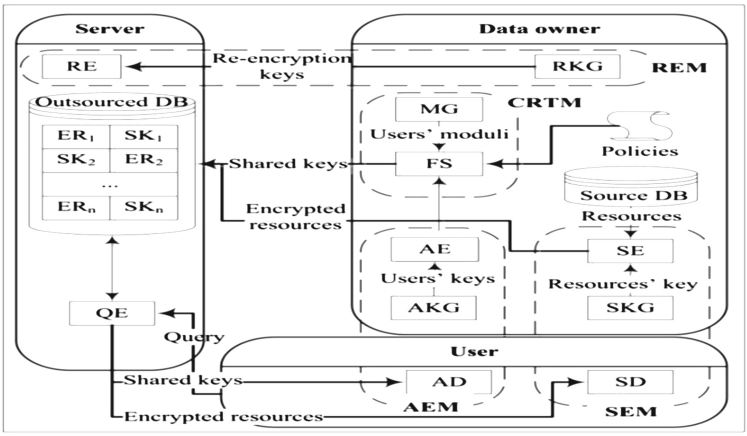

Fig. 2. The architecture of our implemented prototype

5.1 Symmetric Encryption Module

Symmetric Encryption Module (SEM) provides a symmetric key cryptosystem to encrypt resources in our system. There are three functions in this module:

1. Symmetric Key Generator (SKG): in this function three random k-bit keys K_1, K_2, and K_3 and a random b-bits key K_X are generated using *RandGen* function. Here, k is the size of key in *Pgen* algorithm and b is the size of each block of original message M in *AONT* algorithm. The resulted key is $K = (K_1, K_2, K_3, K_X)$, which is used in SE function, SD function, and REM.

2. Symmetric Encryption (SE): the function encrypts message M using the keys K_1, K_2, K_3, and K_X. Algorithm 1 shows the *SE* algorithm.
3. Symmetric Decryption (SD): the function decrypts ciphertext C using the keys K_1, K_2, K_3, and K_X. The SD algorithm is shown in Algorithm 2.

5.2 Asymmetric Encryption Module

The aim of Asymmetric Encryption Module (AEM) is to provide a public-key cryptosystem used for sharing the keys of resources. It contains three functions including, Asymmetric Key Generator (AKG), Asymmetric Encryption (AE), and Asymmetric Decryption (AD). The module can be implemented using a public-key cryptosystem such as RSA.

5.3 Re-Encryption Module

Re-Encryption Module (REM) consists of two functions to provide a proxy re-encryption mechanism for access control policy updates:

1. Re-encryption Key Generator (RKG): the function generates re-encryption keys sent to the server when policy is updated. It is implemented using Algorithm 3.
2. Re-Encryption (RE): the function is used when the data owner needs to update his policies by re-encrypting the cipher-text with the new key. Details of this function are shown in Algorithm 4.

Algorithm1. SE algorithm

Input:	keys K_1, K_2, K_3, and K_X, message $M = m_0, m_1, \ldots, m_{s-1}$
Output:	ciphertext$C = c_0, c_1, \ldots, c_{n-1}$
1.	$M' = AONT(M)$ $(M' = m'_0, \ldots, m'_{n-1})$
2.	$P_1 = PGen(K_1, b)$
3.	$P_2 = PGen(K_2, b)$
4.	$P_3 = PGen(K_3, n)$
5.	$M' = Perm(P_3, M')$
6.	$c_0 = Perm(P_1, m'_0) \oplus Perm(P_2, K_X)$
7.	From i = 1 to n-1: 7.1 $c_i = Perm(P_1, m'_i) \oplus Perm(P_2, c_{i-1})$
8.	Return C

Algorithm 2. SD algorithm

Input:	keys K_1, K_2, K_3, and K_X, ciphertext$C = c_0, c_1, \ldots, c_{n-1}$
Output:	message $M = m_0, m_1, \ldots, m_{s-1}$
1.	$P_1 = PGen(K_1, b)$
2.	$P_2 = PGen(K_2, b)$
3.	$P_3 = PGen(K_3, n)$
4.	From i = n-1 down to 1: 4.1 $m'_i = DePerm\left(P_1, c_i \oplus Perm(P_2, c_{i-1})\right)$
5.	$m'_0 = DePerm(P_1, c_0 \oplus Perm(P_2, K_X))$
6.	$M' = DePerm(P_3, M')$ $(M' = m'_0, \ldots, m'_{n-1})$
7.	$M = AONT^{-1}(M')$
8.	Return M

Algorithm 3. RKG algorithm

Input:	keys K_1, K_2, K_3, and K_X
Output:	re-encryption key REK
1.	$K_1' = RandGen(k)$
2.	$K_2' = RandGen(k)$
3.	$K_3' = RandGen(k)$
4.	$K_X' = RandGen(b)$
5.	$CK_1 = FindCK(PGen(K_1, b), PGen(K_1', b))$
6.	$CK_3 = FindCK(PGen(K_3, n), PGen(K_3', n))$
7.	$REK = (CK_1, CK_3, K_X, K_X', K_2, K_2')$
8.	Return REK

Algorithm **4**. RE algorithm

Input	re-encryption key REK=$(CK_1, CK_3, K_X, K_X', K_2, K_2')$, ciphertext$C = c_0, c_1, \ldots, c_{n-1}$
Output	ciphertext$C' = c_0', c_1{}', \ldots, c_{n-1}{}'$
1.	$P_2 = PGen(K_2)$
2.	$P_2' = PGen(K_2')$
3.	From i = n-1 down to 1:
	3.1 $c_i' = Perm\big(CK_1, c_i \oplus Perm(P_2, c_{i-1})\big)$
4.	$c_0' = Perm\big(CK_1, c_0 \oplus Perm(P_2, K_X)\big)$
5.	$C' = Perm(CK_3, C')$
6.	$c_0' = c_0' \oplus Perm(P_2{}', K_X{}')$
7.	From i=1 to n-1:
	7.1 $c_i{}' = c_i' \oplus Perm(P_2{}', c_{i-1}')$
8.	Return C'

5.4 Chinese Remainder Theorem Module

Chinese Remainder Theorem Module (CRTM) is used by the data owner for key management. It contains two functions:

1. Modulus Generator (MG): for every user u_i, this function generates a modulus n_{s_i} such that $n_{s_1}, n_{s_2}, \ldots, n_{s_i}$ are pairwise relatively prime. These moduli will be used as CRT moduli.
2. Find Solution (FS): the function is used for computing the solution of simultaneous congruences in (1).

6 Theoretical and Experimental Analysis

Three main requirements for an access control enforcement mechanism in the DAS model are as follows:

1. The mechanism requires keeping a few numbers of secret keys by each user. Generally, mechanisms in which users keep a small set of secret keys are more feasible and manageable than those in which the number of keys is unlimited.

2. Efficiency of operations, especially policy updates should be acceptable. In fact, data outsourcing should not lead to unacceptable computational and storage overheads for the data owner.
3. Privacy of access control policy should be preserved. We assume that the server is honest-but-curious. The server may get some information about the content of data, if access control mechanism reveals access control policies.

We show that our proposed solution addresses these requirements, appropriately.

6.1 Number of User's Secret Keys

In our approach, each user needs to keep only a key pair (secret and public keys) to access all of her authorized resources. This advantage is a result of using CRT to compute a shared key value for each resource (x_{K_r}). Thereby, the authorized user can compute the decryption key of the resource and easily access the content of the resource. The second column of Table 1 compares our proposed mechanism with some other known solutions from this point of view. Compared to the proposals in [2, 11], and [12], in our approach the user does not need to derive lots of keys to access her permitted resources. So, accessing the resources is more efficient here and does not need several interactions with the server to drive proper keys.

Table 1. Comparison of our mechanism with some other solutions

Mechanism	Number of keys for each user	Are access policies kept confidential?
Key-derivation based mechanism [1]	More than one (based on the users hierarchy and ACLs)	Yes
Two-layer mechanism [2]	Two	No
Two-layer mechanism with encrypted tokens [14]	Two	Yes
Two-layer mechanism [15]	More than one (based on user membership in the groups)	Yes
Proposed mechanism	One pair (public and secret key)	Yes

6.2 Efficiency of Operations

In a typical selective encryption based access control enforcement mechanism, changing policy may need the update of some resources' encryption keys. Therefore, these resources should be re-encrypted using new encryption keys. Reaching this purpose necessitates conducting the three steps of receiving the resource from the server, decrypting it with the old key and re-encrypting it with the new key, and finally, sending the encrypted resource to the server.

In addition, data owner should inform the users about the key changes. Clearly in such a case, a lot of computational overheads are imposed to the data owner.

Our mechanism transfers a large portion of such computations to the server relying upon the CRT and re-encryption together. Let us investigate the computational cost of grant and revoke operations for the data owner in our solution.

Grant: as we discussed before, a data owner should find the new shared key as the solution of (10) and send it to the server to be replaced by the old one.

Revoke: re-encryption key generation and finding CRT solution x'_{k_r}, according to the new ACL(r), are two main operations in a revoke process. According to the analysis in [10], re-encryption key generation needs two operations of finding conversion key (*FindCK*), four permutation key generations (*PGen*), and four random bits generations (*RandGen*). *FindCK* is cheap and *PGen* and *RandGen* are linear to a symmetric cipher operation. Thus, the re-encryption key generation function is linear to a symmetric cipher operation. On the other hand, computing the CRT solution, using an efficient algorithm suggested in [13], takes $O(tk^2)$ bit operations, where k is the number of bits of each modulus n_{u_i}, t is the number of users in $ACL(r)$, and kt is the number of bits for the CRT solution in (5). Now, we investigate on the relation between the number of users, the required bits for n_{u_i}, and the CRT solution.

Recall that a user u_i has its own modulus n_{u_i} and if $ACL(r) = \{u_1, u_2, \ldots, u_k\}$, then $0 \leq x_{K_r} < n_{u_1} n_{u_2} \ldots n_{u_k}$. On the other hand, users' moduli are co-primes. Therefore, the maximum number of users in a system with m bits for each modulus n_{u_i} is equal to the number of prime numbers between 0 and 2^m. For a resource r whose ACL is the set U of all users, the maximum number of bits required for x_{K_r} is $m * l$ where $l = |U|$. Table 2 represents the maximum number of users and required bits for x_{K_r} based on some possible number of bits of a modulus. We use the Prime Number Theorem which proves that the number of prime numbers less than an integer n is approximately equal to $\frac{n}{\ln n}$. Therefore, finding the CRT solution in (5) takes approximately $O(k * k * \frac{2^k}{lnk})$ bit operations where k is the size of modulus in bit. For instance, in a system with about 5900 users, finding CRT solution takes $O\left(16 * 16 * \frac{2^{16}}{ln16}\right)$ bit operations, in worse case, that is not an expensive process for the data owner.

Table 2. The relation between the number of users and the imposed storage cost of storing $\boldsymbol{x_{K_r}}$

Size of user modulus	Maximum number of users	Size of x_{K_r} (in bits)
8 bit	$(2^8/\ln 2^8) = 46$	8 * 46
16 bit	$(2^{16}/\ln 2^{16}) = 5909$	16 * 5909
32 bit	$(2^{32}/\ln 2^{32}) = 193635335$	32 * 193635335

6.3 Confidentiality of Access Control Policies

In our approach confidentiality of security policies is preserved against both the server and users. When the server receives a request, it sends the cipher-text and the shared key of the requested resources to the requester. However, the server cannot understand whether the user can decrypt the resource or not. Therefore, users' privileges are not

revealed in the query processing scenario. Moreover, policy change in our approach does not reveal access policies to the server.

6.4 Experimental Results

We performed two series of experiments. We ran our programs on Java 2 Standard Edition (J2SE) 1.6.0 and Windows 7 with an Intel(R) Core i5 2.5 GHz processor and 6 GB of main memory. In both series, we used RSA as a public-key cryptosystem.

At first, we evaluated the efficiency of finding a CRT solution in terms of the required time for the data owner with different ACL sizes for a resource. Such a metric allows us to estimate the load on the data owner in the initialization phase as well as for the policy update. The graph in Fig. 3 illustrates the time of finding a CRT solution for a set of congruence equations. We observe that the growth is somehow linear to the size of ACL.

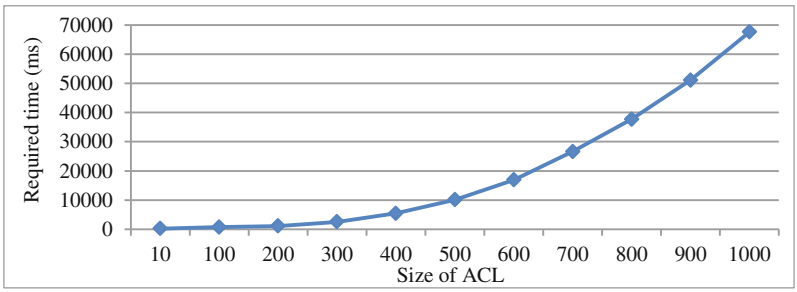

Fig. 3. Required time to find a CRT solution

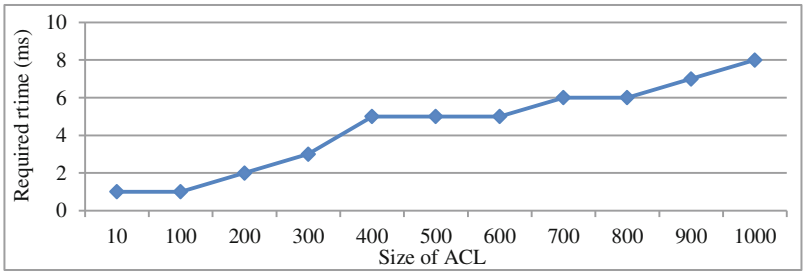

Fig. 4. Required time to drive a resource encryption key

Then, we evaluated the performance of calculating a resource key in terms of the required time for a user to derive the decryption key from a shared key (Fig. 4). Such a metric allows us to estimate the load on the user in the query processing phase. We observe in Fig. 4 that the growth is nearly linear to the size of ACL. Moreover, the time remains low for a reasonable size of ACL, e.g., 8 ms for 1000 users in an ACL.

7 Conclusions

In this paper, we address the problem of enforcing access control policies in database outsourcing scenario. Our mechanism uses Chinese remainder theorem and proxy re-encryption that results in a limited number of users' keys, the efficiency of policy changes, and the protection of access control policies from the untrusted server.

References

1. Damiani, E., di Vimercati, S.D.C., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Key management for multi-user encrypted databases. In: Proceedings of the 2005 ACM Workshop on Storage Security and Survivability, pp. 74–83 (2005)
2. di Vimercati, S.D.C., Jajodia, S., Foresti, S., Paraboschi, S., Samarati, P.: Over-encryption: management of access control evolution on outsourced data. In: VLDB, pp. 123–134 (2007)
3. di Vimercati, S.D.C., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Encryption policies for regulating access to outsourced data. ACM Trans. Database Syst. **35**(2), 1–46 (2010)
4. Tian, X., Wang, X., Zhou, A.: DSP re-encryption: a flexible mechanism for access control enforcement management in daaS. In: Proceedingsof IEEE International Conference on Cloud Computing, pp. 25–32 (2009)
5. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Trans. Inf. Syst. Secur. **9**(1), 1–30 (2006)
6. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005)
7. Wang, G., Liu, Q., Wu, J.: Achieving fine-grained access control for secure data sharing on cloud servers. Concurr. Comput. Pract. Exp. **23**(12), 1443–1464 (2011)
8. Kong, Y., Seberry, J., Getta, J.R., Yu, P.: A cryptographic solution for general access control. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 461–473. Springer, Heidelberg (2005)
9. Tourani, P., Hadavi, M.A., Jalili, R.: Access control enforcement on outsoured data ensuring privacy of access control policies. In: 2011 International Conference on High Performance Computing and Simulation (HPCS), pp. 491–497 (2011)
10. Syalim, A., Nishide, T., Sakurai, K.: Realizing proxy re-encryption in the symmetric world. In: Abd Manaf, A., Zeki, A., Zamani, M., Chuprat, S., El-Qawasmeh, E. (eds.) ICIEIS 2011, Part I. CCIS, vol. 251, pp. 259–274. Springer, Heidelberg (2011)
11. Damiani, E., di Vimercati, S.D.C., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Selective data encryption in outsourced dynamic environments. In: Proceedings of the Second International Workshop on Views on Designing Complex Architectures, pp. 127–142
12. Zych, A., Petković, M., Jonker, W.: Efficient key management for cryptographically enforced access control. Comput. Stand. Interfaces **30**(6), 410–417 (2008)
13. Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC, Boca Raton (1996)
14. di Vimercati, S.D.C., Foresti, S., Jajodia, S., Paraboschi, S., Pelosi, G., Samarati, P.: Preserving confidentiality of security policies in data outsourcing. In: Proceedings of the 7th ACM Workshop on Privacy in the Electronic Society, pp. 75–84 (2008)
15. Lanovenko, A., Guo, H.: Dynamic group key management in outsourced databases. In: Proceedings of the World Congress on Engineering and Computer Science, USA (2007)