The ISC Int'l Journal of Information Security

ISeCure

July 2018, Volume 10, Number 2 (pp. 129–139)

http://www.isecure-journal.org

Enforcing RBAC Policies over Data Stored on Untrusted Server☆

Naeimeh Soltani[1], Ramin Bohlooli[1], and Rasool Jalili[1]

[1] *Department of Computer Engineering Sharif University of Technology Tehran, Iran*

ARTICLE INFO.

Article history:

Received: 5 April 2018
First Revised: 29 May 2018
Last Revised: 23 June 2018
Accepted: 11 July 2018
Published Online: 15 July 2018

Keywords:
Access Control, Outsourced Data, Role-Based Access Control, Chinese Remainder Theorem.

ABSTRACT

One of the security issues in data outsourcing is enforcement of the data owner's access control policies. This includes some challenges. The first challenge is preserving confidentiality of data and policies. One of the existing solutions is encrypting data before outsourcing which brings new challenges; namely, the number of keys required to access authorized resources, efficient policy updating, write access control enforcement, overhead of accessing/processing data at the user/owner side. Most of the existing solutions address only some of challenges, while imposing high overhead on both owner and users. Though, policy management in the Role-Based Access Control (RBAC) model is easier and more efficient due to the existence of role hierarchical structure and role inheritance; most of the existing solutions address only enforcement of policies in the form of access control matrix. In this paper, we propose an approach to enforce RBAC policies on encrypted data outsourced to a service provider. We utilize Chinese Remainder Theorem for key management and role/permission assignment. Efficient user revocation, efficient role hierarchical structure updating, availability of authorized resources for users of new roles, and enforcement of write access control policies as well as static separation of duties, are of advantages of the proposed solution.

© 2018 ISC. All rights reserved.

1 Introduction

Due to reducing communication costs and increasing storage space and flexibility in resource management, Database-as-a-Service (DBaaS) model, in which an organization outsources its data to a service provider (SP), becomes a popular paradigm. Though data outsourcing helps to reduce cost and complexity in the maintenance and management of data, it introduces new security challenges because data is out of the owner's control and also service providers are assumed to be honest-but-curious. They are honest in executing protocols, but curious to learn information about the stored and exchanged data. The easiest way to overcome this issue is to encrypt data before outsourcing.

Because of storing encrypted data on SPs, access control policy enforcement faces some challenges, namely the number of keys needed to access authorized resources, distributing required keys among authorized users, efficient policy updating, write access control enforcement, overhead of accessing data at user side, overhead of processing data at owner side, and preserving confidentiality of data and policies. Most of the existing solutions address only enforce-

☆ An earlier version of this paper has already been presented in ISCISC2017.

* Corresponding author.

Email addresses: nsoltani@ce.sharif.edu (N. Soltani), rbohlooli@ce.sharif.edu (R. Bohlooli), jalili@sharif.edu (R. Jalili)

ISSN: 2008-2045 © 2018 ISC. All rights reserved.

ment of policies in the form of access control matrix, while in the Role-Based Access Control model (RBAC), policy management is easier and more efficient due to the existence of role hierarchical structure and role inheritance.

According to the prevalence use of RBAC model in most organizations, we propose an approach to enforce RBAC policies on encrypted data stored on external SPs. We use Chinese Remainder Theorem (CRT) for key management and role/permission assignment. We support efficient revocation of role membership from a user and update of role hierarchical structure. Also, users of newly added roles have access to all authorized resources without the need to re-encrypt resources. Enforcement of write access control policies and static separation of duties (SSD) are of advantages of the proposed approach.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 describes our proposed approach in detail. Section 4 analyzes our solution and describes its advantages. Finally, Section 5 concludes the paper.

2 Related Work

Existing approaches in enforcement of access control policies enforce policies either in the form of access control matrix or in the RBAC model.

Solutions of the first group have used tree-based key derivation techniques ([1, 2]), attribute-based encryption ([3–5]), and CRT ([6, 7]) to preserve confidentiality of data and policies and also to distribute each resource encryption key among authorized users. Solutions of tree-based key derivation techniques are not scalable as producing unique keys for users and resources is a serious bottleneck. Solutions using attribute-based encryption are scalable but inefficient. They are scalable, because each user is assigned a key derived from his attributes which allows him to access an arbitrary number of resources, and they are inefficient due to using bilinear map that increases computational overhead.

CP-ABE [1] is a variation of ABE which is used in several schemes (e.g. [8], [9]). In CP-ABE, a set of attributes are associated with the decryption key and the access policy. A decryption key can decrypt a ciphertext only if it's associated attributes satisfy the access policy of the ciphertext[8]. A trusted authority sends or distributes these attributes to intended users for decryption purpose which also allow access to shared data by having these required attributes [10]. In [8] a CP-ABE was proposed which introduced a scalable immediate revocation scheme, named SPIRC [2]. The proposed scheme in [9] separates the duty of enforcing access control policy from enforcing constraint policy to enhance security. This scheme also defined and used the "Sensitive Data Set Constraint" concept to prevent conflict of interests.

For efficient user revocation and reducing the overhead of policy update at owner side, proxy re-encryption, over-encryption, and lazy revocation are used in such solutions. In proxy re-encryption [3, 4, 11] data owner produces a new key and delivers it to the SP to re-encrypt the data. The concept of over-encryption was proposed in [12]. This approach uses two-layer encryption. The first layer is done by the owner to provide initial protection. The second layer is done by the service provider and enables fine-grained access control to data. When a user is revoked, the new encryption key of the second layer would be sent to the server to re-encrypt this layer. In [13] an effective design and implementation of this approach is presented with detailed design choices. The idea behind the lazy revocation [14] is, since the user had access to the old data before eviction, it can be assumed that the user had cached that data, therefore it is not important to re-encrypt old data. Based on this idea, if the user's access is revoked from data, the data are not re-encrypted immediately. Instead, only the new data are encrypted with a new version of the key so that the evicted user can no longer access the new data. In CRT-based solutions which use CRT for key management, update of policies is equivalent to add/remove an equation to/from the CRT equation set.

Enforcement of write access control policies [5, 15, 16] is another challenge. A proposed solution to this challenge is granting a token to authorized users. This token would be used by the users to prove their write permission to the SP. Also, signing the resources by the owner and authorized users after each write operation is used to prevent unauthorized write and to guarantee the integrity of data.

In the second group focusing on RBAC, an approach is to use Hierarchical ID-based in which identities are organized in a hierarchical structure. A key associated to an identity can decrypt resources encrypted to itself as well as its descendant identities, but cannot decrypt resources intended for other identities. In the case of any hierarchy change, the identities and their associated keys may need to be changed. In addition, the size of the keys increases corresponding to the growth in the hierarchical structure of identities [17, 18]. Attribute-based encryption has similar problems [19].

[1] Ciphertext-Policy Attribute-Based Encryption

[2] Scalable Proxy-based Immediate Revocation

Asghar *et al.* [20] proposed a different solution in which policies and user's requests are encrypted and no common key exists among users. So, user revocation is done without re-encryption.

In 2011, Zhou *et al.* [21] proposed an access control scheme based on Identity-based broadcast encryption (IBBE), which suffers from changing the specification of all roles when a user is revoked. In IBBE, a broadcaster sends the encrypted message for a set of users based on their identities.

The same authors in 2013[22] proposed a role-based encryption (RBE) scheme that achieves efficient user revocation. They present an RBE-based hybrid cloud storage architecture that allows an organization to store data securely in a public cloud, while maintaining the sensitive information such as role hierarchical structure, in a private cloud. In this architecture, decryption of resources is done in two steps; the first step is done at the cloud and the second step is done at the user side, so the overhead of accessing the resources would be increased. In addition, in order to enable users of newly added roles to access the authorized resources, re-encryption is needed. In another paper in 2015, Zhou *et al.* [23] introduced two more RBE schemes. In one of the schemes, they focused on efficient user revocation and in the other, they proposed an approach to enable users of newly added roles have access to all authorized resources.

In [24] a new approach, called ACV-BGKM [3], was proposed which uses linear algebra based approach on a structure called Access Control Vector (ACV)[25]. Consuming massive computational and storage resources for frequent updates of the group key are disadvantages of this approach[26].

In 2015 SathiyaBalan *et al.* [25] identified the applicability of the CRT based Group Key Management (CRTGKM) for cloud environment. It is reported that the scheme is more computationally efficient than the other cloud environment group key management schemes known as ACV-BGKM and AB-GKM which require high computation time for key update and key recovery processes.

3 Proposed Approach

Assigning permissions to roles and roles to users are of key features in RBAC model. In our approach, a secret key is associated to each role and resource, which is transparent to users and provides efficiency of user revocation. In "permission assignment", each resource secret key is shared among authorized roles and in "role assignment", each role secret key is shared among authorized users, using CRT. An agent located at the user side acts as the interface between the user and SP. Updating policies and role hierarchy is done with minimal overhead and users of newly added roles have access to all authorized resources without re-encryption. Our approach also supports the enforcement of SSD and write access control policies.

Before describing the details of the proposed approach, we briefly introduce Chinese Remainder Theorem as our key management method.

3.1 Chinese Remainder Theorem and key management

Chinese remainder theorem states that for the system of simultaneous congruence in (1), where $k \geq 2$, the positive integers $n_1, n_2, ..., n_k$ are pairwise relatively prime, and $(n_1, n_2, ..., n_k) \in Z$, there exists a unique solution x, such that $x \in [0, \prod_{i=1}^{k} n_i)$.

$$\begin{cases} x \equiv a_1 \mod n_1 \\ x \equiv a_2 \mod n_2 \\ ... \\ x \equiv a_k \mod n_k \end{cases} \tag{1}$$

According to (2), we can share a secret key between privileged users $(u_1, u_2, ..., u_k)$ in which K is the secret key, E is the encryption algorithm, x_K is the shared key and n_{u_i} is module assigned to the user i.

$$\begin{cases} x_K \equiv E_{K_{u_1}}(K) \mod n_{u_1} \\ x_K \equiv E_{K_{u_2}}(K) \mod n_{u_2} \\ ... \\ x_K \equiv E_{K_{u_k}}(K) \mod n_{u_k} \end{cases} \tag{2}$$

Granting access of key K to a new user u_{k+1} is equivalent to adding an equation to the equation set (2) and calculating the new shared key x' in the equation set (3).

$$\begin{cases} x'_K \equiv x \mod n_{u_1} n_{u_2} ... n_{u_k} \\ x'_K \equiv E_{K_{u_{k+1}}}(K) \mod n_{u_{k+1}} \end{cases} \tag{3}$$

Also for revoking access of user u_k from the key K, we should remove an equation from the equation set (2) and solve x'' in the relation (4).

$$x''_K \equiv x \mod n_{u_1} n_{u_2} ... n_{u_{k-1}} \tag{4}$$

3.2 Proposed Approach Components

In our proposed architecture, four components exist: owner, user, agent, and SP. In addition, roles and

[3] Access Control Vector Broadcast Group Key Management

resources should also be considered. The description of these components are as follows:

- **Resources**: Resources contain everything that the owner wants to outsource to the SP. For each resource r_i, these properties are defined by the owner:
  - ID_{r_i}: A unique identifier,
  - $(Kpub_{r_i}, Kpriv_{r_i})$: A public/private key pair,
  - SK_{r_i}: A secret key.

  The $Kpriv_{r_i}$ acts as the write token and the SK_{r_i} is used for encrypting the resource. Both of these keys are shared among authorized roles using CRT to create separate shared keys for read and write access control. The write token is also used by the owner and authorized users for signing the resource after each write operation.
- **Roles**: A role is a set of access permissions. In the RBAC model, roles are defined in a hierarchical structure to ease defining new roles and permission assignment/revocation. Figure 1, shows an example of a role hierarchy in which permissions of roles decrease by moving from up to down in the hierarchical structure. The arrows are directed from ancestors to descendants. In addition, the focus is on limited role hierarchies in which a role may have one or more direct ancestors, but is restricted to a single direct descendant. In this example, the role A has the most permissions and the roles B and E are called its direct ancestors and the roles C and D are its indirect ancestors. The role A does not have any descendant and the roles A, B, and C are all descendants of the role D. Roles A and B are called indirect descendants of the role D and the role C is its direct descendant. For each role like l_i, the following properties are defined:
  - ID_{l_i}: A unique identifier,
  - SK_{l_i}: A secret key,
  - m_{l_i}: The CRT module of the role,
  - P_{l_i}: The "public parameter" of the role which is defined as the set of secret keys of ancestor roles, encrypted by the role's secret key (SK_{l_i}).

  The SK_{l_i} and m_{l_i} are used in "permission assignment" process in which read or write access of a resource is granted to a role using CRT. Existence of P_{l_i} allows users of a role to inherit permissions of ancestor roles.
- **Users**: A user is anyone who wants to access the resources. Properties defined for the user u_i are:
  - ID_{u_i}: A unique identifier,
  - n_{u_i}: The CRT module of the user,

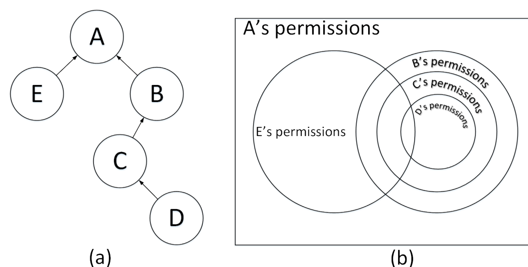


Figure 1. Roles hierarchy (a) and inheritance of permissions (b)

  - $(Kpub_{u_i}, Kpriv_{u_i})$: A public/private key pair.

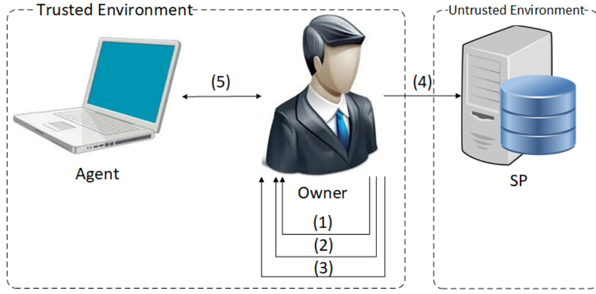
  $Kpub_{u_i}$ and n_{u_i} are used in the "role assignment" process in which a role is assigned to the user. $Kpriv_{u_i}$ is used to resolve the secret key of a role (SK_{l_i}) assigned to the user, from role's shared key.
- **SP**: An honest-but-curious service provider which stores encrypted information and serves the user via the agent installed at the user side. The agent acts as an interface between the user and the SP and prevents unauthorized users from direct access to the SP and stored data.
- **Owner**: The original owner of the resources. Owner produces required IDs, keys, and modules, encrypts resources and assigns permissions to roles and roles to users and also sends encrypted data to the SP. We also define the following properties for the owner:
  - K_{OA}: A shared key between the owner and agent(s),
  - n_O: The CRT module of the owner,
  - $(Kpub_O, Kpriv_O)$: The public/private key pair of the owner,
  - SK_O: The secret key of the owner.

  Although the last three properties are not mandatory, we can use them to eliminate the necessity of storing all (roles' and resources') keys at the owner side as described in Section 3.4 and Section 3.5.
- **Agent**: A software agent which would be executed in the users' machines. This component receives requests from the user, communicates with SP and returns the response to the user. In the other words, the agent makes the system transparent for users.

3.3 Overall view

There are two main phases in this approach:

- **Outsourcing data**: This phase would be done by the owner when the system is launched. In this procedure, the owner generates required keys, IDs, and modules as well as encrypting



(1) Generating IDs, keys and modules
(2) Initial role/permission assignment
(3) Encrypting resources
(4) Outsourcing data
(5) Share a key with the agent

Figure 2. Outsourcing data

Table 1. Users

ID	Roles	Module	Public Key
$E_{K_{OA}}(ID_{u_1})$	$E_{K_{OA}}(ID_A)$	$E_{K_{OA}}(n_{u_1})$	$E_{K_{OA}}(Kpub_{u_1})$
$E_{K_{OA}}(ID_{u_2})$	$E_{K_{OA}}(ID_B)$ $E_{K_{OA}}(ID_E)$	$E_{K_{OA}}(n_{u_2})$	$E_{K_{OA}}(Kpub_{u_2})$
$E_{K_{OA}}(ID_{u_3})$	$E_{K_{OA}}(ID_B)$	$E_{K_{OA}}(n_{u_3})$	$E_{K_{OA}}(Kpub_{u_3})$
...	...	...	...

Table 2. Resources

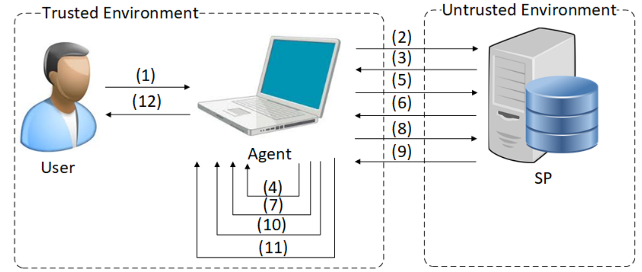
ID	Resource	Shared Keys	Roles	Public Key
$E_{K_{OA}}(ID_R)$	$E_{SK_R}(R)$	xr_R xw_R	$E_{K_{OA}}(ID_E)$ $E_{K_{OA}}(ID_D)$	$E_{K_{OA}}(Kpub_{r_i})$
...	...	...	...	...

Table 3. Roles

ID	Public Parameter	Shared Key	Module
$E_{K_{OA}}(ID_A)$	$E_{SK_A}(SK_B\|ID_B)$ $E_{SK_A}(SK_E\|ID_E)$	x_A	$E_{K_{OA}}(m_A)$
$E_{K_{OA}}(ID_B)$	$E_{SK_B}(SK_C\|ID_C)$	x_B	$E_{K_{OA}}(m_B)$
$E_{K_{OA}}(ID_C)$	$E_{SK_C}(SK_D\|ID_D)$	x_C	$E_{K_{OA}}(m_C)$
$E_{K_{OA}}(ID_D)$	-	x_D	$E_{K_{OA}}(m_D)$
$E_{K_{OA}}(ID_E)$	-	x_E	$E_{K_{OA}}(m_E)$

the resources, defining roles hierarchy, performing initial assignments (roles to users and permissions to roles), pushing all required information on the SP and shares a key with the agent (K_{OA}). Figure 2 represents these steps. Some examples of information stored on the SP for users, resources and roles are represented in Table 1, 2 and 3, respectively.

The owner can update role hierarchy and change permissions after the outsourcing phase with minimal overhead.



(1) request read or write access of a resource as a specific role
(2),(3) Retrieve roles of the user
(4) verify the user against the claimed role
(5),(6) retrieve shared key of the role with the direct access to the requested resource
(7) Extract the secret key of the role
(8),(9) Retrieve the (encrypted) resource and its shared key
(10) Extract the secret key of the resource
(11) Decrypt the resource
(12) Response (decrypted resource)

Figure 3. Processing queries

- **Processing user requests**: This phase contains the routine operation of the system. Users send their requests to their agents. The agent then communicates the information about the user, his role and the requested resource with the SP and extracts required keys. Finally, if the user is authorized, the agent returns the decrypted resource to the user. Figure 3 shows the steps of processing a user's request (more details are discussed in Section 3.6).

The details of operations that should/could be done in the above phases are described in the rest of this section.

3.4 Role Assignment

Assigning a role to a user means sharing the role's secret key with the user. In order to assign role A to users $u_1, u_2, ..., u_k$, the owner goes through the following steps:

(1) Encrypting SK_A with $Kpub_{u_i}$ for $1 \leq \ i \ \leq k$.
(2) Forming the equation set (5). Solution of the equation set, x_A, would be the *shared key* of the role A.

$$\begin{cases} x_A \equiv E_{Kpub_{u_1}}(SK_A) \mod n_{u_1} \\ ... \\ x_A \equiv E_{Kpub_{u_k}}(SK_A) \mod n_{u_k} \\ x_A \equiv E_{Kpub_O}(SK_A) \mod n_O \end{cases} \tag{5}$$

As pointed earlier, the last equation is used to to eliminate the owner from storing secret key of the role.

(3) Storing x_A in table ROLES on SP.

If authorized user u_i wants to access a resource as a users of the role A, the agent requests x_A from the SP and extract K_A using relations (6) and (7). This key would be used later to retrieve the decryption key of the available resources.

$$E_{Kpub_{u_i}}(SK_A) \equiv x_A \mod n_{u_i} \quad (6)$$

$$SK_A = D_{Kpriv_{u_i}}(E_{Kpub_{u_i}}(SK_A)) \quad (7)$$

Assigning the role A to a new user u_{k+1} is equivalent to re-organize the CRT equation set with a new equation corresponding to u_{k+1} (equation set (8)) and solve it again to achieve a new shared key (x'_A). However, this could be done more efficient by solving the equation set (9).

$$\begin{cases} x'_A \equiv E_{Kpub_{u_1}}(SK_A) \mod n_{u_1} \\ \ldots \\ x'_A \equiv E_{Kpub_{u_k}}(SK_A) \mod n_{u_k} \\ x'_A \equiv E_{Kpub_{u_{k+1}}}(SK_A) \mod n_{u_{k+1}} \\ x'_A \equiv E_{Kpub_O}(SK_A) \mod n_O \end{cases} \quad (8)$$

$$\begin{cases} x'_A \equiv x_A \mod n_{u_1} \ldots n_{u_k} n_o \\ x'_A \equiv E_{Kpub_{u_i}}(SK_A) \mod n_{u_{k+1}} \end{cases} \quad (9)$$

Revocation of the user u_i is equivalent to form the equations set without the ith equation and solve that to obtain a new shared key.

In most real applications, there are some (static) constraints on the set of roles which could be assigned to a specific user. These constraints are handled via "static separation of duties" (SSD). Before assigning roles to users, the owner defines a set of rules to define SSD and uses these rules to specify the set of available roles for each user. In order to assign a role to a user, the owner selects the role with the highest authority from the aforementioned set.

3.5 Permission Assignment

Permission assignment consists of granting read or write access of a resource to a set of roles. If the owner wants to grant read permission to a set of roles, the secret key of the resource is shared among authorized roles with the lowest authority using CRT. For example, to assign read permission on resource R to all roles in Figure 1 (A, B, C, D, E) the following steps would be done:

(1) The "minimal covering set" of given roles should be calculated; which includes all roles that either have no ancestor or none of their ancestors is in the given set. In our example, the covering set would be $\{E, D\}$.

Table 4. Public parameter of roles in Figure 1

Role	Public Parameter
A	$E_{SK_A}(SK_B \vert ID_B), E_{SK_A}(SK_E \vert ID_E)$
B	$E_{SK_B}(SK_C \vert ID_C)$
C	$E_{SK_C}(SK_D \vert ID_D)$
D	-
E	-

(2) The owner forms the CRT equation set (10) with secret keys of the obtained roles in step (1). The solution of the equation set, xr_R is the shared key of the resource and would be stored on the SP (in table RESOURCES).

$$\begin{cases} xr_R \equiv E_{SK_D}(SK_R) \mod m_D \\ xr_R \equiv E_{SK_E}(SK_R) \mod m_E \\ xr_R \equiv E_{SK_O}(SK_R) \mod m_O \end{cases} \quad (10)$$

(3) Users of the roles E or D can access the resource R via their agents, directly.

Users of other authorized roles gain the resource secret key using the public parameter of their roles which contains encrypted keys of ancestor roles. Table 4 indicates the public parameter of roles in Figure 1 where $(SK_i|ID_i)$ shows the secret key and ID of the role i $(i \in \{A, B, C, D, E\})$ concatenated together and E is the symmetric encryption algorithm.

Granting write permission is also done in the same way, by sharing the resource write token among authorized roles with the lowest authority using CRT (equation set (11)).

$$\begin{cases} xw_R \equiv E_{SK_D}(Kpriv_R) \mod m_D \\ xw_R \equiv E_{SK_E}(Kpriv_R) \mod m_E \\ xw_R \equiv E_{SK_O}(Kpriv_R) \mod m_O \end{cases} \quad (11)$$

As the final stage of permission assignment, the resource is signed and encrypted by the owner, using the resource write token (resource private key) and the resource secret key, respectively. Signing the resource is done due to integrity check and preventing SP from unauthorized write. Authorized users and the owner can verify the signature and sign the resource after each change, using the resource's write token given from the write shared key.

Revoking permissions of the resource R from a role is equivalent to removing an equation from the equation set (10) and calculating the new shared key.

3.6 Policy Enforcement

Let's assume that the required information about users, resources, and roles are stored as Table 1, 2,

and 3 at SP. Among this information, some of them are encrypted by the key K_{OA} which is shared between the owner and the agent and makes encrypted information accessible to the agent, too.

Based on this assumption, each time a user asks for a resource as a role, the agent checks if the user belongs to the specified role or not using Table 3 and 1. If the user is correct, the agent will ask the shared key of the user claimed role from the owner and extract the role's secret key as illustrated in relations (6) and (7) using the private key of the user. Then, a record containing user ID, role ID, role's secret key, and timestamp of the role's shared key will be added to "current sessions" table stored at the agent, for future uses.

Timestamp shows the creation time of the role shared key and is stored to prevent the agent from downloading and extracting duplicate shared keys.

To make it clear, let's continue with an example. If sample user U_3 sends a request containing (ID_{U_3}, ID_B, ID_R, n_{U3}) to the agent in order to ask for reading the resource R as a user with role B, the agent goes through the following steps:

(1) The agent checks if the user U_3 belongs to the users of role B or not. In order to do that, requests encrypted role ID of the user U_3 from SP and according to the Table 1 receives $E_{K_{OA}}(ID_B)$ as response. As illustrated before, the key K_{OA} is shared between the owner and the agent. So the agent decrypts $E_{K_{OA}}(ID_B)$ and gains ID_B which is the same as the role ID claimed by the user. So, the agent accepts the user's request.

(2) In the second step, the agent requests encrypted role IDs with direct access to the resource R and according to the Table 2 receives $E_{K_{OA}}(ID_E)$ and $E_{K_{OA}}(ID_D)$ from the SP. After decrypting the two encrypted IDs, gains ID_E and ID_D which are not the same as the user's role ID, ID_B; In this situation, the users of the role B can access the resource R if and only if the role B inherits permissions of the role D or the role E. In the other words, if and only if, the role B is a direct or indirect descendant of roles D or E. So the agent checks permissions of the direct and indirect ancestors of the role B using its public parameter. The role C is the direct ancestor of the role B. As the role C does not have direct access to the resource R, the agent searches for permissions of the ancestor of the role C. This process continues till the agent find a path in the role hierarchical structure of Figure 1 from the role B to one of the roles with direct access to the resource R (roles D or E). If the agent could not find any path, the user's request is denied.

(3) To gain ancestor of the role B, the agent requests public parameter and shared key of the role B from the SP. SP sends $E_{SK_B}(SK_C|ID_C)$ and x_B as response to the agent. Then the agent retrieves the secret key of the role B, (SK_B) from x_B using private key and module of the user U3. Finally using SK_B, the agent gets SK_C and ID_C which are the secret key and ID of the role C.

(4) Because of no equality between ID_C and IDs of roles with direct access to the resource (ID_E, ID_D), the agent requests ancestor of the role C and receives $E_{SK_C}(SK_D|ID_D)$ from the SP. After decrypting $E_{SK_C}(SK_D|ID_D)$ gains SK_D and ID_D which are secret key and ID of the role D that has direct read access to the resource R.

(5) Finally, using the secret key and public parameter of the role B, the agent finds the secret key of the role D which has direct access to the resource R. Then asks for encrypted resource and its read shared key along with module of the role D. After receiving $E_{SK_R}(R)$, xr_R, and m_D from the SP, gains SK_R according to the relations (12) and (13). Using the key SK_R, now the agent decrypts the resource.

$$xr_R \equiv E_{SK_D}(SK_R) \mod m_D \tag{12}$$

$$SK_R = D_{SK_D}(E_{SK_D}(SK_R)) \tag{13}$$

Note that the agent gets the secret key of the resource R after $n+1$ role key decryption where n is the number of roles between the user's role and the role with direct access to the resource.

3.7 Adding a Role to Role Hierarchy

In order to add a new role to the hierarchical structure, first of all, we should create its public parameter from role IDs of its direct ancestors. Secondly, the public parameter of the descendant role should be updated. For example, by adding the role P between the roles A and B in Figure 1, the role A becomes the direct descendant of the role P and the role B becomes its direct ancestor.

As the ancestor of the role A changes from the role B to the role P, so the public parameter of the role A should be updated. As a result $E_{SK_A}(SK_B)$ will be replaced with $E_{SK_A}(SK_P)$. Actually, it is not necessary to remove $E_{SK_A}(SK_B)$, because after adding the role P, the role B changes to indirect ancestor of the role A and the role A still inherits permissions of the role B, but $E_{SK_A}(SK_P)$ should be added to persist role inheritance. Also, the secret key of the role P will be encrypted with the secret key of the role A and the secret key of the role B will be re-encrypted

with the secret key of the role P. In the other words, a record containing ($E_{K_{OA}}(ID_P)$, $E_{SK_P}(SK_B)$, x_P, $E_{K_{OA}}(m_P)$) will be added to Table 3 and it is worth noted that users of newly added role have access to all authorized resources without need to the resource re-encryption.

3.8 Removing a Role from Role Hierarchy

In order to remove a role such as C, the related record in Table 3 is removed and then the secret key of the removed role's direct ancestor will be re-encrypted with the secret key of the removed role's descendant. In this example, $E_{SK_B}(SK_C)$ is replaced with $E_{SK_B}(SK_D)$ to show that the role D is ancestor of the role B. At the end, if there is any resource that the role C has had direct access to it, the shared key of the resource will be changed by removing the equation from the CRT equation set of the resource's shared key.

4 Evaluation

In this section, we evaluate the proposed solution in terms of different criteria. In some criteria, we use experimental results to prove the efficiency of our approach. The experimental results are gained from a Java application ran in Windows 8 with an Intel(R) Core(TM) i7-4700HQ 2.40 GHz CPU and 8G RAM. We used 128-bit keys with AES algorithm for encryption of roles and resources keys and also 1024 bit keys with RSA algorithm for creating the shared key of each role. The modules for roles and users are all 1024 bits.

The evaluation metrics are as follow:

- Policy confidentiality: All information stored at SP including IDs and keys for roles, users, and resources are encrypted.
- Scalability: Let us first consider the maximum number of supported users. As illustrated, in our approach "role assignment" and "permission assignment" is based on CRT. Each user is given a public/private key pair and a CRT module. Modules are pairwise relatively prime numbers. Therefore, the maximum number of users in a "role assignment" equation set with m bits modulus is equal to the number of prime numbers between 0 and 2^m. Prime Number Theorem proves that the number of prime numbers less than an integer n is approximately equal to $\frac{n}{\ln n}$. As a result the maximum number of equations in a CRT equation set with m bit modulus is approximately equal to $\frac{2^m}{m \times \ln 2}$ and there will be no bottleneck in increasing the number of roles, users, and resources; but as discussed, the value of x in the equation set (1) is in the range $[0, \prod_{i=1}^{k} n_i)$. Therefore, an increase in the length of the module causes a growth in the length of the shared key.

Table 5. Relation Between Module Size and Number of Users and Shared Key Size

User Module Size	Maximum No. Of Users	Shared Key Size
8 bit	46	8 * 46
16 bit	5909	16 * 5909
32 bit	193635335	32 * 193635335

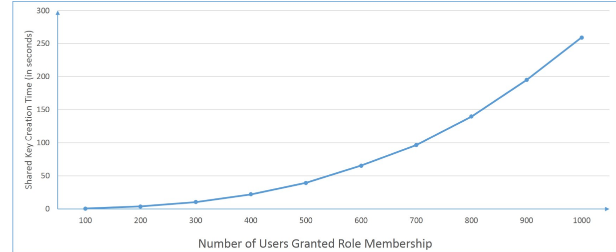


Figure 4. Role shared key creation time in conjunction with number of users

The size of the ciphertext is another important factor in scalability. In our solution, encrypted resources or ciphertexts and their encryption keys do not contain any information related to the roles and users. Therefore, the length of the secret key for each resource is constant and ciphertext size is linearly proportional to the size of plaintext. However, in Zhou *et al.* [22] approach, the resource key is produced according to the list of the roles that have access to the resource and if the resource size exceeds one megabyte, the time needed to produce encryption key will equal with the time needed for resource encryption and decryption.
- Computational overhead at the user side: User side operations contain resolving the secret key of a role, and secret key and write token of a resource from their shared keys. Figure 4 depicts creation time of a shared key for a specific role and Figure 5 shows the required time for resolving the role key from its shared key according to the number of users. Totally, the access time to a resource is $O(L)$ where L is the number of all roles in the system (in the worst case, all roles should be traversed sequentially to reach the role with direct access to the resource). It is clear that the operation time on the user side is minimal. This factor is not evaluated in [20] and [22].
- The efficiency of policy updates: In Section 3, we discussed add/remove a role to/from the role hierarchical structure and grant/revoke role membership to/from a user. In all these oper-

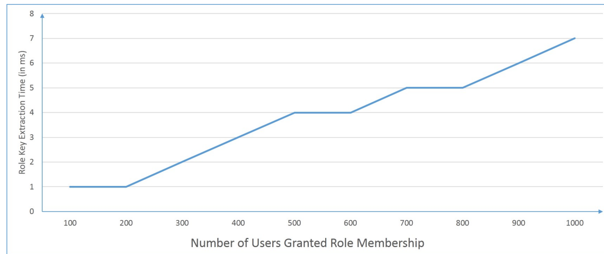


Figure 5. Role secret key extraction time in conjunction with number of users

ations, there is no need to create new secret keys for roles and resources or re-encrypting the resources. As our focus is on limited role hierarchies in which a role may have one or more direct ancestors but is restricted to a single direct descendant, in "add role" operation, only one role key re-encryption is needed in addition to creating a recode for the new role containing encrypted role ID, encrypted form of ancestor's role key, and role shared key. In the "remove role" operation, the public parameter of the descendant of the removed role should be updated to contain the encrypted secret key of the ancestor of the removed role. So time complexity of role hierarchical structure is O(1).

Grant/revoke role membership to/from a user is simply done by adding/removing an equation to/from CRT equation set and creating a new shared key for the role. In addition, due to updating information of Table 1 after grant or revoke, if the revoked user asks for accessing the resources as the specified role, the agent will refuse his request; Therefore, the owner may ignore creation of a new shared key for the role after revoking the user and delays new shared key creation to a time that number of revoked users becomes high.

- Availability of resources for users of new roles: In our approach, resource key is only shared among authorized roles with the lowest authority and other roles access the keys using their "public parameter". Therefore, users of newly added roles can access to the authorized resources using "public parameter" of their role, but in Zhou *et al.* solution [22], re-encryption is required after adding a new role to the roles hierarchy.
- Enforcement of static separation of duties: As illustrated in Section 3 user's information is kept in Table 1 which contains IDs of the roles assigned to each user and the agent use these role IDs to check correctness of the user claimed role when he/she wants to access a resource. These role IDs are also useful for enforcing rules of static separation of duties. The owner checks the list of roles assigned to a user each time he wants to assign a new role to the user.
- Enforcement of write access control policies: Enforcing write access control policies is done using a write token assigned to each resource. This write token is also used for integrity check.

Table 6. Comparison of our approach with some other solutions

	Asghar *et al.* [20]	Zhou *et al.* [22]	Proposed solution
Ciphertext size is independent of the number of roles and users	√	√	√
Scalability	√	√	√
Using role hierarchical structure	√	√	√
Using role inheritance in authorization	√	√	√
Accessibility of resources for new role users	√	×	√
Enforcement of static separation of duties	×	×	√
Enforcement of write access control policies	×	×	√
Overhead of adding/removing a role	not mentioned	not mentioned	low
Overhead of revoking role membership from user	low	low	low

Table 6 depicts our proposed approach compared to other solutions in this criteria.

5 Discussion and Conclusion

Nowadays, enforcement of access control policies over data stored on an untrusted server has raised significant security issues. In this paper, we proposed an approach for enforcement of access control policies in the RBAC model. In the proposed approach, role/permission assignment and key management are based on CRT that results in eliminating storage overhead on the owner, scalability, efficient user revocation, and efficient policy updates. By policy update, we mean grant/revoke role membership to/from a user and adding/removing a role to/from role hierarchy. It is worth noting that by adding a new role to the role hierarchical structure, users of newly added roles have access to all authorized resources and there is no need to resource re-encryption. Also, enforcement of write access control policies and static separation of duties are considered in our solution.

In this paper, we discussed enforcement of write

access control policies in general, using a write token (resource's private key) and sharing it between authorized roles, but write operation on a resource can be interpreted as insert, delete, or update. In order to differentiate these three permissions, we can use one separate token and the corresponding shared key for each of them that results in an increase in the storage and computational overhead. As future work, we will investigate a way to enforce these permissions with higher efficiency and lower storage overhead.

Examining lattice-based role hierarchies in which a role may have more than one descendants is another trend for future work. Based on the present approach using a lattice-based role hierarchy increases the computational overhead of role hierarchy updates.

One more trend for future work is eliminating unauthorized users and SP inferring useful information from encrypted data which is exchanged between the agent and SP.

References

[1] Ernesto Damiani, S. De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Key management for multi-user encrypted databases. In *Proceedings of the 2005 ACM Workshop on Storage Security and Survivability*, StorageSS '05, pages 74–83, New York, NY, USA, 2005. ACM.

[2] D. Grolimund, L. Meisser, S. Schmid, and R. Wattenhofer. Cryptree: A folder tree structure for cryptographic file systems. In *Reliable Distributed Systems, 2006. SRDS '06. 25th IEEE Symposium on*, pages 189–198, Oct 2006.

[3] Junbeom Hur and Dong Kun Noh. Attribute-based access control with efficient revocation in data outsourcing systems. *Parallel and Distributed Systems, IEEE Transactions on*, 22(7):1214–1221, 2011.

[4] T. Eissa and Gi-Hwan Cho. A fine grained access control and flexible revocation scheme for data security on public cloud storage services. In *Cloud Computing Technologies, Applications and Management (ICCCTAM), 2012 International Conference on*, pages 27–33. IEEE, Dec 2012.

[5] Rohollah Mahfoozi. Using functional encryption to manage encrypted data. Master's thesis, Computer Engineering Department, Tehran, IRAN: Sharif University of Technology, November 2013.

[6] P. Tourani, M.A. Hadavi, and R. Jalili. Access control enforcement on outsourced data ensuring privacy of access control policies. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 491–497. IEEE, July 2011.

[7] Leila Karimi, Seyyed Ahmad Javadi, Mohammad Ali Hadavi, and Rasool Jalili. *Missing a Trusted Reference Monitor: How to Enforce Confidential and Dynamic Access Policies?*, pages 92–104. Springer International Publishing, Cham, 2014.

[8] Divyashikha Sethia, Huzur Saran, and Daya Gupta. Cp-abe for selective access with scalable revocation: A case study for mobile-based healthfolder. *IJ Network Security*, 20(4):689–701, 2018.

[9] Nurmamat Helil and Kaysar Rahman. Cp-abe access control scheme for sensitive data set constraint with hidden access policy and constraint policy. *Security and Communication Networks*, 2017, 2017.

[10] Samta Ukey, Jayant Adhikari, et al. A review on data storage security in cloud computing environment for mobile devices. *International Journal of Research*, 5(13):312–316, 2018.

[11] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. Ieee, IEEE, march 2010.

[12] Sabrina De Capitani Di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Over-encryption: management of access control evolution on outsourced data. In *Proceedings of the 33rd international conference on Very large data bases*, pages 123–134. VLDB endowment, 2007.

[13] Enrico Bacis, Sabrina De Capitani di Vimercati, Sara Foresti, Stefano Paraboschi, Marco Rosa, and Pierangela Samarati. Access control management for secure cloud storage. In *International Conference on Security and Privacy in Communication Systems*, pages 353–372. Springer, 2016.

[14] Rohit Jain and Sunil Prabhakar. *Access Control and Query Verification for Untrusted Databases*, pages 211–225. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[15] Sabrina De Capitani Di Vimercati, Sara Foresti, Sushil Jajodia, Giovanni Livraga, Stefano Paraboschi, and Pierangela Samarati. Enforcing dynamic write privileges in data outsourcing. *Computers and Security*, 39, Part A(0):47 – 63, 2013. 27th {IFIP} International Information Security Conference.

[16] Lanju Kong, Qingzhong Li, and Lin Li. Enabling access control in partially honest outsourced databases. *International Journal of Database Theory and Application*, 7(3):63–72, 2014.

[17] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In *Advances in Cryptology ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 548–566. Springer Berlin Heidelberg, 2002.

[18] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *Advances in Cryptology EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456. Springer Berlin Heidelberg, 2005.

[19] Yan Zhu, Di Ma, Chang-Jun Hu, and Dijiang Huang. How to use attribute-based encryption to implement role-based access control in the cloud. In *Proceedings of the 2013 International Workshop on Security in Cloud Computing*, Cloud Computing '13, pages 33–40. ACM, 2013.

[20] Muhammad Rizwan Asghar, Mihaela Ion, Giovanni Russello, and Bruno Crispo. Espoonerbac: Enforcing security policies in outsourced environments. *Computers and Security*, 35(0):2 – 24, 2013. Special Issue of the International Conference on Availability, Reliability and Security (ARES).

[21] Lan Zhou, Vijay Varadharajan, and Michael Hitchens. Enforcing role-based access control for secure data storage in the cloud. *Computer*, 54(10):1675–1687, 2011.

[22] L. Zhou, V. Varadharajan, and M. Hitchens. Achieving secure role-based access control on encrypted data in cloud storage. *IEEE Transactions on Information Forensics and Security*, 8(12):1947–1960, Dec 2013.

[23] Lan Zhou, Vijay Varadharajan , and Michael Hitchens. Generic constructions for role-based encryption. *International Journal of Information Security*, 14(5):417–430, Oct 2015.

[24] Mohamed Nabeel, Ning Shang, and Elisa Bertino. Privacy preserving policy-based content sharing in public clouds. *IEEE Transactions on Knowledge and Data Engineering*, 25(11):2602–2614, 2013.

[25] V. K. SathiyaBalan, P. Zavarsky, D. Lindskog, and S. Butakov. Study of applicability of chinese remainder theorem based group key management for cloud environment. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 114–119, Dec 2015.

[26] Perumal Pandiaraja, Pandi Vijayakumar, Varadarajan Vijayakumar, and Raman Seshadhri. Computation efficient attribute based broadcast group key management for secure document access in public cloud. *J. Inf. Sci. Eng.*, 33(3):695–712, 2017.

Naeimeh Soltani received her B.S. degree in information technology engineering from Isfahan University of Technology in 2013 and her M.S. degree in information technology from Sharif University of Technology in 2015. Her research interests includes data outsourcing and network security.

Ramin Bohlooli received his B.S. degree in computer engineering from Amirkabir University of Technology (a.k.a Tehran's Polytechnique) in 2014. He received his M.S. degree in computer engineering from Sharif University of Technology in 2018. His research interests includes data outsourcing and searchable encryption on which he has worked since 2014.

Rasool Jalili received his B.S. degree in computer science from Ferdowsi University of Mashhad in 1985, and M.S. degree in computer engineering from Sharif University of Technology in 1989. He received his Ph.D. in computer science from University of Sydney, Australia, in 1995. He then joined the department of computer engineering, Sharif University of Technology in 1995. He has published more than 140 papers in international journals and conference proceedings. He is now an associate professor, doing research in the areas of computer dependability and security, access control, distributed systems, and database systems in his Data and Network Security Laboratory (DNSL).