Achieving Secure Role-based Access Control on Encrypted Data in Cloud Storage

Lan Zhou,Vijay Varadharajan,and Michael Hitchens

***Abstract*—With the rapid developments occurring in cloud computing and services, there has been a growing trend to use the cloud for large-scale data storage. This has raised the important security issue of how to control and prevent unauthorised access to data stored in the cloud. One well-known access control model is the role-based access control (RBAC), which provides flexible controls and management by having two mappings, users to roles and roles to privileges on data objects. In this paper, we propose a role-base encryption (RBE) scheme which integrates the cryptographic techniques with RBAC. Our RBE scheme allows RBAC policies to be enforced for the encrypted data stored in public clouds. Based on the proposed scheme, we present a secure RBE based hybrid cloud storage architecture which allows an organisation to store data securely in a public cloud, while maintaining the sensitive information related to the organisation's structure in a private cloud. We describe a practical implementation of the proposed RBE based architecture, and discuss the performance results. We demonstrate that users only need to keep a single key for decryption and system operations are efficient regardless of the complexity of the role hierarchy and user membership in the system.**

***Index Terms*—Role-based Access Control, Data Storage, Role-based Encryption, Cloud Computing, Architecture.**

I. Introduction

There has been a growing trend in the recent times to store data in the cloud with the dramatic increase in the amount of digital information such as consumers' personal data to larger enterprises wanting to back up databases or store archival data. Cloud data storage can be particularly attractive for users (individuals or enterprises) with unpredictable storage demands, requiring an inexpensive storage tier or a low-cost, long-term archive. By outsourcing users' data to the cloud, service providers can focus more on the design of functions to improve user experience of their services without worrying about resources to store the growing amount of data. Cloud can also provide on demand resources for storage which can help service providers to reduce their maintenance costs. Furthermore, cloud storage can provide a flexible and convenient way for users to access their data from anywhere on any device.

However, several recent surveys [1], [2] show that 88% potential cloud consumers are worried about the privacy of their data, and security is often cited as the top obstacle for cloud adoption. There are different types of infrastructures associated with a cloud [3]. A public cloud is a cloud which is made available to the general public, and resources are allocated in a pay-as-you-go manner. A private cloud is an internal cloud that is built and operated by a single organisation. The organisation has full control of the private cloud, and the private cloud cannot be accessed by external parties. Hence a private cloud is often considered to be more secure and trusted. A recent survey [4] shows that nearly half, 43% of all companies report utilising private clouds and 34% of companies say they will begin to use some form of private cloud in the next six to twelve months.

In this paper, we address the issue of secure data storage in the public cloud. Public cloud is formed by one or more data centres often distributed geographically in different locations. Users do not know where their data is stored and there is a strong perception that users have lost control over their data after it is uploaded to the cloud. In order to allow users to control the access to their data stored in a public cloud, suitable access control policies and mechanisms are required. The access policies must restrict data access to only those intended by the data owners. These policies need to be enforced by the cloud. In many existing cloud storage systems, data owners have to assume that the cloud providers are trusted to prevent unauthorised users from accessing their data.

In role-based access control (RBAC) model, roles are mapped to access permissions and users are mapped to appropriate roles. For instance, users are assigned membership to the roles based on their responsibilities and qualifications in the organisation. Permissions are assigned to qualified roles instead of individual users. Moreover, in RBAC, a role can inherit permissions from other roles, hence there is a hierarchical structure of roles. Since being first formalised in 1990's, RBAC has been widely used in many systems to provide users with flexible access control management, as it allows access control to be managed at a level that corresponds closely to the organisation's policy and structure.

In traditional access control systems, enforcement is carried out by trusted parties which are usually the service providers. In a public cloud, as data can be stored in distributed data centres, there may not be a single central authority which controls all the data centres. Furthermore the administrators of the cloud provider themselves would be able to access the data if it is stored in plain format. To protect the privacy of the data, data owners employ cryptographic techniques to encrypt the data in such a way that only users who are allowed to access the data as specified by the access policies will be able to do so. We refer to this approach as a policy based encrypted data access. The authorised users who satisfy the access policies will be able to decrypt the data using their private key, and no one else will be able to reveal the data content. Therefore, the problem of managing access to data stored in the cloud is transformed into the problem of management of keys which in turn is determined by the access policies.

In this paper, we present the design of a secure RBAC-based cloud storage system where the access control policies are enforced by a new role-based encryption (RBE) that we proposed in the paper. This RBE scheme enforces RBAC policies on encrypted data stored in the cloud with an efficient user revocation using an broadcast encryption[1] mechanism described in [5]. In our RBE scheme, the owner of the data encrypts the data in such a way that only the users with appropriate roles as specified by a RBAC policy can decrypt and view the data. The role grants permissions to users who qualify the role and can also revoke the permissions from existing users of the role. The cloud provider (who stores the data) will not be able to see the content of the data if the provider is not given the appropriate role. Our RBE scheme is able to deal with role hierarchies, whereby roles inherit permissions form other roles. A user is able to join a role after the owner has encrypted the data for that role. The user will be able to access that data from then on, and the owner dose not need to re-encrypt the data. A user can be revoked at any time in which case, the revoked user will not have access to any future encrypted data for this role. With our new RBE scheme, revocation of a user from a role does not affect other users and roles in the system. In addition, we outsource part of the decryption computation in the scheme to the cloud, in which only public parameters are involved. By using this approach, our RBE scheme achieves an efficient decryption on the client side. We have also used the same strategy of outsourcing to improve the efficiency of the management of user to role memberships, involving only public parameters.

Based on the proposed RBE scheme, we develop a secure cloud data storage architecture using a hybrid cloud infrastructure. This hybrid cloud architecture is a composite of private cloud and public cloud, where the private cloud is used to store only the organisation's sensitive structure information such as the role hierarchy and user membership information, and the public cloud is used to store the actual data that is in the encrypted form. The high level architecture of the hybrid cloud storage system is illustrated in Fig. 1. In this architecture, the users who wish to share or access the data only interact with the public cloud; there is no access for public users to access the private cloud, which greatly reduces the attack surface for the private cloud. This architecture not only dispels the organisation's concerns about risks of leaking sensitive structure information, but also takes full advantage of public cloud's power to securely store large volume of data. Another significant benefit of this architecture is that it overcomes collusion attacks such as the public cloud colluding with a revoked user, thereby allowing this user to decrypt data that has been encrypted to a role of which the user was a member previously.

We have developed a secure cloud storage system using the new RBE scheme and hybrid cloud architecture. The most frequently used system operations such as encryption of data by a data owner, decryption of data by a cloud user have been benchmarked. The result shows that the encryption and decryption time for a given data size is constant regardless of the number of roles and users that have the access to the data. Since part of the decryption computation is outsourced to the cloud, the cloud's decryption time increases with the growth in the number of users in the role (to which data has been encrypted). We have optimised the implementation of the decryption algorithm and show that the cloud's decryption time can be reduced by increasing the processor cores. Hence when deployed in a cloud, depending on the scale of the system, our architecture can be tailored to achieve the desired response time by adjusting the number of virtual processor cores. In addition, we have used certain specific characteristics of elliptic curves to further improve the performance of the computations. By making use of these features, our new RBE architecture achieves a more efficient and practical secure cloud data storage system.

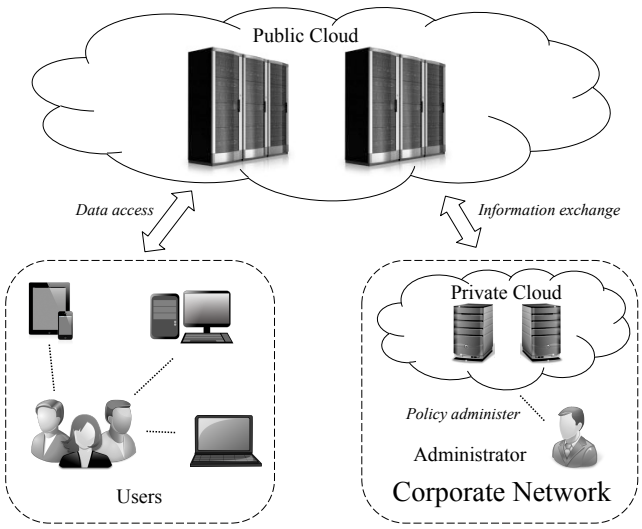


Fig. 1: Hybrid Storage Cloud

Main Contributions: The main contributions of this paper are (i) a new role-based encryption (RBE) scheme with efficient user revocation that combines RBAC policies with encryption to secure large scale data storage in a public cloud, (ii) a secure RBAC based hybrid cloud storage architecture which allows an organisation to store data securely in a public cloud, while maintaining the sensitive information related to the organisation's structure in a private cloud, (iii) a practical implementation of the proposed RBE scheme and description of its architecture and (iv) analysis of results demonstrating efficient performance characteristics such as efficient encryption and decryption operations on the client side as well as superior characteristics of the proposed RBE scheme such as constant size ciphertext and decryption key as well as efficient user revocation. Given these characteristics, the proposed RBE system has the potential to be a suitable candidate for developing practical commercial cloud data storage systems.

The rest of this paper is organised as follows. Section II discusses some related works and compares them with our new RBE scheme. The preliminaries of our new RBE scheme is described in Section III. In Section IV, we introduce our new RBE scheme. The architecture for our secure cloud storage

[1]A broadcast encryption scheme is an encryption scheme where messages can be encrypted and securely broadcast to a group of users who are listening in a broadcast channel.

TABLE I: Comparison of Schemes for Enforcing RBAC Policies

	HKM	HIBE	ABE[2]	RBE in [6]	RBE in [7]	Our New Scheme
Constant size ciphertext	√	√	—	√	—	√
Constant size keys when in a single role	√	—	—	√	√	√
Constant size keys when in multiple roles	√	—	—	√	—	√
User management delegation	—	—	—	√	—	√
Revoking user not affecting other users/roles	—	—	—	—	—	√
No need of re-encryption after user revocation	—	—	—	—	—	√

system is presented in Section V. Section VI describes the methods that are used in our implementation, and gives the result of our experiments. Section VII concludes the paper.

II. Related Work

One approach to enforce access control policies is to transform the access control problem into a key management problem. In the literature, there exist many hierarchy access control schemes [8]–[10] which have been constructed based on hierarchical key management (HKM) schemes, and approaches using HKM schemes to enforce RBAC policies for data storage are discussed in [11]–[13]. However, these solutions have several limitations. For instance, if there is a large number of data owners and users involved, the overhead involved in setting up the key infrastructure can be very high indeed. Furthermore, when a user's access permission is revoked, all the keys known to this user as well as all the public values related to these keys need to changed, which makes these schemes impractical.

An alternative approach for the management of keys is Hierarchical ID-based Encryption (HIBE), such as [14], [15]. However, in a HIBE scheme, the length of the identity becomes longer with the growth in the depth of hierarchy. In addition, the identity of a node must be a subset of its ancestor node so that its ancestor node can derive this node's private key for decryption. Therefore, this node cannot be assigned as a descendant node of another node in the hierarchy tree unless the identity of the other role is also the super set of this node's identity.

Recently we have seen the development of schemes built directly on RBAC policies. We introduced a role-based encryption scheme (RBE) in [6]. However, the user revocation in this scheme requires the update of all the role related parameters. Another scheme was proposed in [7]. In this scheme, the size of the ciphertext increases linearly with the number of all the predecessor roles. In addition, if a user belongs to different roles, multiple keys need to be possessed by this user. Moreover, the management of the user membership for each individual role requires the use of the system secret keys. The scheme proposed in this paper overcomes these limitations, and each role can use its own secret keys to manage the user membership without the need to know the system secret keys. Moreover, the scheme proposed in this paper provides efficient user revocation.

[2]The term "ABE" here refers to the approaches that use ABE schemes to enforce RBAC policies, not the ABE schemes themselves.

Besides RBAC, there are also other access control models such as Attribute Based Access Control (ABAC). In ABAC, access is granted based on attributes of the user. Systems define combination of attributes as the access policies, and users need to prove that they have these attributes in order to gain access. In 2006, the first attribute-based encryption (ABE) scheme was proposed in [16] based on the work in [17], and some other ABE schemes have been proposed afterwards. In these schemes, data is encrypted to a set of attributes, and users who have the private keys associated with these attributes can decrypt the data. These works have provided an alternative approach to secure the data stored in a distributed environment using a different access control mechanism, such as [18]. In [6], we have shown that an ABE scheme can be used to enforce RBAC policies. However, in that approach, the size of user key is not constant, and the revocation of a user will result in a key update of all the other users of the same role. [19] also investigated the solutions of using ABE scheme in RBAC model. However their solution only maps the attributes to the role level in RBAC, and they assumed that the RBAC system itself would determine the user membership.

Other approaches to protect data privacy in a cloud environment include using direct encryption and proxy re-encryption. In these cryptographic schemes, data is allowed to be encrypted directly to the users with whom the owners wish to share the data [20], [21]. This is analogous to the access control policies in Discretionary Access Control (DAC) model. Hence they are usually used in systems where DAC model is adopted. Since the permissions in such systems are specified either in a flat out structure or in a access matrix, we do not compare them with our schemes as the access policies are specified differently in RBAC model.

The comparison of the existing schemes which can be used to enforce RBAC policies and our new scheme proposed in this paper is shown in Table I.

III. Preliminaries

A. Role-based Encryption Systems

Our RBE scheme has the following four types of entities. SA is a system administrator that has the authority to generate the keys for users and roles, and to define the role hierarchy. RM is a role manager[3] who manages the user membership

[3]In systems where there are small number of users, the SA can act as the role manager to manage the user membership of each role to make the systems compact. However, in large scale systems, it is infeasible for a single party to manage all the users, therefore separate role managers make the user management more flexible and efficient.

of a role. Owners are the parties who want to store their data securely in the cloud. Users are the parties who want to access and decrypt the stored data in the cloud. Cloud is the place where data is stored and it provides interfaces so all the other entities can interact with it.

We define the following algorithms for our RBE scheme:

Setup (λ) takes as input the security parameter λ and outputs a master secret key mk and a system public key pk. mk is kept secret by the SA while pk is made public to all users of the system.

Extract (mk, ID) is executed by the SA to generate the key associated with the identity ID. If ID is the identity of a user, the generated key is returned to the user as the decryption key. If ID is the identity of a role, the generated key is returned to the RM as the secret key of the role, and an empty user list $\mathcal{RUL}$ which will list all the users who are the members of that role is also returned to the RM.

ManageRole (mk, ID_R, $\mathcal{PR}_R$) is executed by the SA to manage a role with the identity ID_R in the role hierarchy. $\mathcal{PR}_R$ is the set of roles which will be the ancestor roles of the role. This operation publishes a set of public parameters pub_R to cloud.

AddUser (pk, sk_R, $\mathcal{RUL}_R$, ID_U) is executed by the role manager RM of a role R to grant the role membership to the user ID_U, which results in the role public parameters pub_R and role user list $\mathcal{RUL}_R$, being updated in cloud.

RevokeUser (pk, sk_R, $\mathcal{RUL}_R$, ID_U) is executed by a role manager RM of a role R to revoke the role membership from a user ID_U, which also results in the role public parameters pub_R and role user list $\mathcal{RUL}_R$, being updated in cloud.

Encrypt (pk, pub_R) is executed by the owner of a message M. This algorithm takes as input the system public key pk, the role public parameters pub_R, and outputs a tuple $\langle C, K\rangle$, where C will be a part of the ciphertext, and $K \in \mathcal{K}$ is the key that will be used to encrypt the message M. (Note the ciphertext consists of C and the encrypted M).

We assume that the system uses a secure encryption scheme Enc, which takes $\mathcal{K}$ as the key space, to encrypt messages. The ciphertext of the message M will be in the form of $\langle C, Enc_K(M)\rangle$ which can only be decrypted by the users who are the members of the role R.

When this operation finishes, a ciphertext is output and uploaded to cloud by the owner.

Decrypt (pk, pub_R, dk, C) is executed by a user who is a member of the role R. This algorithm takes as input the system public key pk, the role public parameters pub_R, the user decryption key dk, the part C from the ciphertext downloaded from cloud, and outputs the message encryption key $K \in \mathcal{K}$. The key K can then be used to decrypt the ciphertext part $Enc_K(M)$ and obtain the message M.

B. The Bilinear Pairings

Let $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ be three cyclic groups of prime order p, and $\mathbb{G}_T$ be a cyclic multiplicative group of prime order p. g and h are two random generators where $g \in \mathbb{G}_1, h \in \mathbb{G}_2$. A bilinear pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ satisfies the following properties:

- *Bilinear:* for $a, b \in \mathbb{Z}_p^*$ we have $\hat{e}(g^a, h^b) = \hat{e}(g, h)^{ab}$.
- *Non-degenerate:* $\hat{e}(g, h) \neq 1$ unless $g = 1$ or $h = 1$.
- *Computable:* the pairing $\hat{e}(g, h)$ is computable in polynomial time.

In this paper, we use an asymmetric bilinear pairing which takes inputs from two distinct isomorphic groups $\mathbb{G}_1$, $\mathbb{G}_2$, so that a wider range of curves is allowed to be used in our system. Assume that an elliptic curve E is defined over a field $\mathbb{F}_q$, then $\mathbb{G}_1$ is a subgroup of points on this elliptic curve denoted by $E(\mathbb{F}_q)$, and $\mathbb{G}_2$ is usually a subgroup of $E(\mathbb{F}_{q^k})$, where k is a parameter called the embedding degree in pairing-based cryptography. The average size of the elements in $\mathbb{G}_2$ is larger than that of the elements in $\mathbb{G}_1$. Therefore the computation in $\mathbb{G}_1$ is faster than in $\mathbb{G}_2$. In this paper, we will make use of this characteristic to improve the performance of our RBE scheme when built from the broadcast encryption scheme in [5].

IV. Role-based Encryption Scheme Construction

In this section, we propose an RBE scheme which is designed using asymmetric bilinear groups described in subsection III-B. This RBE scheme is used in our cloud storage system to enforce the RBAC policies.

A. Our Role-based Encryption construction

Now we describe the RBE scheme as follows,

Setup: Generate three groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, and a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Randomly choose two generators $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$, two secret values $s, k \leftarrow \mathbb{Z}_p^*$ and two hash functions $H_1 : \{0,1\}^* \to \mathbb{Z}_p^*$, $H_2 : \mathbb{G}_T \to \mathbb{G}_1$. The master secret key mk and system public key pk are defined as

$$\mathsf{mk} = (s, k, h), \quad \mathsf{pk} = (w, v, w^s, g^k, g, g^s, \cdots, g^{s^q})$$

where $w = h^s, v = \hat{e}(g, h)$, and q is the maximum number of users that each role can have and the maximum depth of the role hierarchy.

Extract(mk, ID)**:** When $\mathsf{ID} = \mathsf{ID}_U$ is an identity of a user U, SA computes the user secret as

$$\mathsf{dk}_U = h^{\frac{1}{s + H_1(\mathsf{ID}_U)}}$$

and gives dk_U to the user U. dk_U is the secret key of the user and it will be used to decrypt the data.

When $\mathsf{ID} = \mathsf{ID}_R$ is an identity of a role R, SA first computes the role secret as

$$\mathsf{sk}_R = g^{\frac{1}{s + H_1(\mathsf{ID}_R)}}$$

and gives sk_R to the role mange of R together with the $\mathcal{RUL}_R$ which is initially set to empty.

ManageRole(mk, ID_R, $\mathcal{PR}_R$)**:** Assume that $\mathcal{PR}_R$ is a set of identities $\{\mathsf{ID}_{R_1}, \cdots, \mathsf{ID}_{R_m}\}$ of all the roles which will be the new ancestor roles of a role R with the identity ID_R. To place this role R in the role hierarchy, SA publishes the

tuple $(A_R, B_R, \mathcal{RUL}_R)$ as role public parameters in the cloud where

$$A_R = h^{(s+H_1(\mathsf{ID}_R))\prod_{i=1}^{m}(s+H_1(\mathsf{ID}_{R_i}))}, \quad B_R = A_R^k$$

AddUser(pk**,** sk_{R_i}**,** $\mathcal{RUL}_{R_i}$**,** ID_{U_k}**):** Assume that the role manager RM of role R_i wants to add a user U_k with the identity ID_{U_k} to the role. $\mathcal{RUL}_{R_i}$ is the set of n users who belong to the role R_i and U_k is not in $\mathcal{RUL}_{R_i}$. The role manager RM first sends the identity ID_{U_k} to the cloud. When receiving the user's identity, the cloud computes the value

$$Y_i = g^{(s+H_1(\mathsf{ID}_{U_k}))\prod_{j=1}^{n}(s+H_1(\mathsf{ID}_{U_j}))}$$

and returns Y_i to the role manager RM. Assume that Y_i' is the existing parameter that RM has received from the cloud previously, and $Y_i' = g$ if Y_i is the parameter that RM receives from cloud for the first time. RM verifies the following equation:

$$\hat{e}(Y_i', w^s \cdot w^{H_1(\mathsf{ID}_{U_k})}) \stackrel{?}{=} \hat{e}(Y_i, w)$$

If the equation holds, RM chooses two random values $r_i, t_i \leftarrow \mathbb{Z}_p^*$ if Y_i is received from cloud for the first time, or uses the existing r_i, t_i otherwise. Then it computes

$$K_i = v^{r_i}, \quad T_i = g^{-t_i}, \quad W_i = w^{-r_i},$$

$$V_i = Y_i^{r_i} = g^{r_i\cdot(s+H_1(\mathsf{ID}_{U_k}))\prod_{j=1}^{n}(s+H_1(\mathsf{ID}_{U_j}))},$$

$$S_i = H_2(K_i)\cdot \mathsf{sk}_{R_i}\cdot g^{kt_i} = H_2(v^{r_i})\cdot g^{\frac{1}{s+H_1(\mathsf{ID}_{R_i})}+kt_i}$$

RM adds ID_{U_k} into $\mathcal{RUL}_{R_i}$, and sends the tuple (T_i, W_i, V_i, S_i) to the cloud. The cloud then publishes another set of role parameters as

$$(\mathsf{ID}_{R_i}, W_i, V_i, S_i, \mathcal{RUL}_{R_i})$$

RevokeUser(pk**,** sk_{R_i}**,** $\mathcal{RUL}_{R_i}$**,** ID_U**):** To revoke a user U_k from a role R_i which has a set $\mathcal{N}$ of n users in $\mathcal{RUL}_{R_i}$, and $U_k \in \mathcal{N}$. The role manager RM first removes ID_{U_k} from role user list $\mathcal{RUL}_{R_i}$ and sends the user identity ID_{U_k} to the cloud. When receiving the user's identity, the cloud computes the value

$$Y_i = g^{\prod_{j=1,j\neq k}^{n}(s+H_1(\mathsf{ID}_{U_j}))}$$

and returns Y_i to the role manager RM. Assume that Y_i' is the existing parameter that RM received from the cloud previously. RM verifies the following equation

$$\hat{e}(Y_i', w) \stackrel{?}{=} \hat{e}(Y_i, w^s \cdot w^{H_1(\mathsf{ID}_{U_k})})$$

If the equation holds, RM chooses two random values $r_i', t_i' \leftarrow \mathbb{Z}_p^*$ and re-computes

$$K_i = v^{r_i'}, \quad T_i = g^{-t_i'}, \quad W_i = w^{-r_i'},$$

$$V_i = Y_i^{r_i'} = g^{r_i'\cdot\prod_{j=1,j\neq k}^{n}(s+H_1(\mathsf{ID}_{U_j}))},$$

$$S_i = H_2(K_i)\cdot \mathsf{sk}_{R_i}\cdot g^{kt_i'} = H_2(v^{r_i'})\cdot g^{\frac{1}{s+H_1(\mathsf{ID}_{R_i})}+kt_i'}$$

RM then replaces the old role parameters (T_i', W_i', V_i', S_i') in the cloud with the new values.

Encrypt(pk**,** pub_{R_x}**):** Assume that the owner of the data M wants to encrypt M for the role R_x. The owner randomly picks $z \leftarrow \mathbb{Z}_p^*$, and computes

$$C_1 = w^{-z}, \quad C_2 = A_{R_x}{}^z, \quad C_3 = B_{R_x}{}^z, \quad K = v^z$$

Then the owner uses K to encrypt the message M, and upload the ciphertext together with $C = \{C_1, C_2, C_3\}$ to the cloud.

Decrypt(pk**,** pub_{R_i}**,** dk_{U_k}**,** C**):** Assume that a role R_x has a set $\mathcal{R}$ of ancestor roles, and the set $\mathcal{M} = R_x \cup \mathcal{R}$ has m roles $\{R_1, \cdots, R_m\}$. $R_i \in \mathcal{M}$ is one ancestor role of R_x, and there is a set $\mathcal{N}$ of n users $\{U_1, \cdots, U_n\}$ in R_i. When a user U_k who is a member of the role R_i wants to decrypt the ciphertext C, the user first requests the ciphertext from the cloud. The cloud computes

$$D = \hat{e}(T_i, C_3),\ g^{p_{i,\mathcal{M}}(s)}, g^{p_{k,\mathcal{N}}(s)},$$

$$Aux_1 = \prod_{j=1,j\neq i}^{m} H_1(\mathsf{ID}_{R_j}),\ Aux_2 = \prod_{j=1,j\neq k}^{n} H_1(\mathsf{ID}_{U_j})$$

where

$$p_{i,\mathcal{M}}(s) = \frac{1}{s}\cdot\Big(\prod_{j=1,j\neq i}^{m}(s+H_1(\mathsf{ID}_{R_j})) - \prod_{j=1,j\neq i}^{m}(H_1(\mathsf{ID}_{R_j}))\Big)$$

$$p_{k,\mathcal{N}}(s) = \frac{1}{s}\cdot\Big(\prod_{j=1,j\neq k}^{n}(s+H_1(\mathsf{ID}_{U_j})) - \prod_{j=1,j\neq k}^{n}(H_1(\mathsf{ID}_{U_j}))\Big)$$

Then the cloud returns the following tuple and the ciphertext of M to the user U_k,

$$(C_1, C_2, D, g^{p_{i,\mathcal{M}}(s)}, g^{p_{k,\mathcal{N}}(s)}, Aux_1, Aux_2)$$

After receiving the ciphertext from the cloud, U_k recovers the systemic encryption key that is used to encrypt M by computing

$$K = (\hat{e}(g^{p_{i,\mathcal{M}}(s)}, C_1)\cdot\hat{e}(S_i\cdot H_2(K_i)^{-1}, C_2)\cdot D)^{\frac{1}{Aux_1}}$$

where

$$K_i = (\hat{e}(V_i, \mathsf{dk}_{U_k})\cdot\hat{e}(g^{p_{k,\mathcal{N}}(s)}, W_i))^{\frac{1}{Aux_2}}$$

By using the key K, U_k can decrypt the ciphertext of M and recover the data M.

Note that the **Decrypt** algorithm requires the expansion of two polynomials of m (the number of ancestor roles) and n (number of users) degrees respectively. This computation could be time consuming if m and n are very large numbers. We note that in computing these two polynomials, no secret values are required. The computation only takes as input system public keys and identities of users and roles. Therefore, outsourcing these computations to the cloud will significantly reduce the workload of users in decryption. The decryption time will also be reduced as the cloud has much more computer power than a user device. Therefore, we have made the cloud compute these two polynomials and pass the results to users in the decryption step. This approach also helps avoid transferring the full identity lists of users and roles which may cause lots of network traffic if the lists are long.

Security Analysis: We have shown that our scheme is semantically secure under the General Decisional Diffie-Hellman

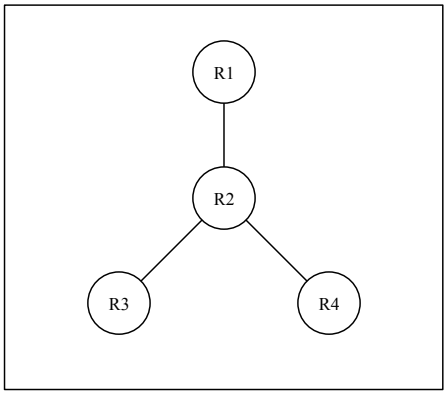


Fig. 2: RBAC Example

Exponent assumption (GDDHE) introduced in [15] by defining a specific GDDHE problem. Security properties and their proofs are provided in Appendix A.

B. Example Scenario

In this subsection, we use a general RBAC example to illustrate our RBE scheme and explain how the proposed scheme supports the role inheritance in decryption.

Let us first look at the RBAC example shown in Fig. 2. Four roles are created in a hierarchical structure. The role R_2 inherits from R_3 and R_4, and R_1 inherits from R_2. Assume that all the required algorithms in the proposed scheme have been executed properly to setup the system parameters. We first look at the case where an owner runs the **Encrypt** algorithm to encrypt a message M to the role R_3. The inputs of the algorithm are the system public keys pk and the role public parameters pub_{R_3} of R_3, and the output of the algorithm is the ciphertext tuple C.

Assume that the role R_1 has a set of user members $\{U_1, U_2, U_3\}$, and the user U_1 wants to access the message M. Since R_1 inherits from R_2, and hence inherits from R_3, the user U_1 is allowed by the policy to access M. U_1 executes the algorithm **Decrypt** to recover the message M, and the inputs of the algorithm are pk, the role public parameters pub_{R_1}, the user decryption key dk_{U_1} and the ciphertext C. The ancestor role set $\mathcal{M}$ has the roles $\{R_1, R_2, R_3\}$, and the user set $\mathcal{N}$ used in the algorithm is $\{U_1, U_2, U_3\}$. Then the algorithm outputs the message M if the decryption key dk_{U_1} that U_1 holds is valid.

Note that users of any role in the set $\mathcal{M}$ can run the **Decrypt** algorithm to decrypt M, and the users do not need to use the role public parameters of the role to which the message was encrypted. Only the role public parameters of the role to which they belong are required in the decryption. From the above described example, we can see how our proposed scheme supports role inheritance in the data decryption.

C. An Extended RBE Scheme for Multiple-role Encryption Support

In the above described RBE scheme, we have shown how to encrypt data to a single role in a RBAC system so that only the authorised users can decrypt the data. In this subsection, we show an extension of our proposed RBE scheme which supports the encryption for multiple roles. In the extended scheme, only the **Encrypt** and **Decrypt** algorithms are modified from the the ones in the original scheme, and the other algorithms are the same as in the original RBE scheme. The two modified algorithms in the extended RBE scheme are described as follows,

Encrypt(pk, $\{\mathsf{pub}_{R_x}\}_{x\in[1,l]}$): Assume that the owner of the data M wants to encrypt M for the set of roles $(R_1, R_2, \ldots, R_l)$. The owner randomly picks $z \leftarrow \mathbb{Z}_p^*$, and computes

$$K = v^z,\ C_1 = w^{-z},\ \{C_{2,i} = {A_{R_i}}^z,\ C_{3,i} = {B_{R_i}}^z\}_{i\in[1,l]}$$

Then the owner uses K to encrypt the message M, and upload the ciphertext together with $C = (C_1, \{C_{2,i}, C_{3,i}\}_{i\in[1,l]})$ to the cloud.

Decrypt(pk, pub_{R_i}, dk_{U_k}, C): Assume that each role $R_x,\ x \in [1, l]$ in the set to which the data is encrypted has an ancestor role set $\mathcal{R}_x$, and a role $R_i \in \mathcal{R}_j$ is an ancestor role of $R_j$where $l \leq j \leq l$. We denote $\mathcal{N}$ as the n user members $\{U_1, \cdots, U_n\}$ of R_i, and $\mathcal{M} = R_j \cup \mathcal{R}_j$. When a user $U_k \in \mathcal{N}$ wants to decrypt the ciphertext C, the user first requests the ciphertext from the cloud. The cloud computes and returns the following tuple and the ciphertext of M to the user U_k in the same way as in the original RBE scheme,

$$(C_1, C_2, D, g^{p_{i,\mathcal{M}}(s)}, g^{p_{k,\mathcal{N}}(s)}, Aux_1, Aux_2)$$

After receiving the ciphertext from the cloud, U_k recovers the systemic encryption key K that is used to encrypt M in the same way as in the original RBE scheme. By using the key K, U_k can decrypt the ciphertext of M and recover the data M.

Compared with the original RBE scheme, the extended RBE scheme inherits most of the features from the original scheme except that the size of the ciphertext is now linearly proportional to the number of roles to which the data is encrypted. However, we note from the above description that the size of the additional parameters required for the encryption for extra roles is relatively small. Therefore, the ciphertext size will remain to the similar level as the size of plaintext which we will show in Section VI.

V. Architecture

In this section, we present the architecture of our secure cloud storage system. It is a hybrid cloud architecture comprising a private cloud which is used to store sensitive role hierarchy of the organisation and user memberships, and a public cloud storing the encrypted data and public parameters associated with the RBE system. The users who wish to access the encrypted data and the data owners who wish to encrypt their data only interact with the public cloud. The role hierarchy and user to role mappings related to the organisation is maintained in the private cloud which is only accessible to the administrator of the organisation. The administrator specifies the role hierarchy and the role managers who manage the user membership relations.

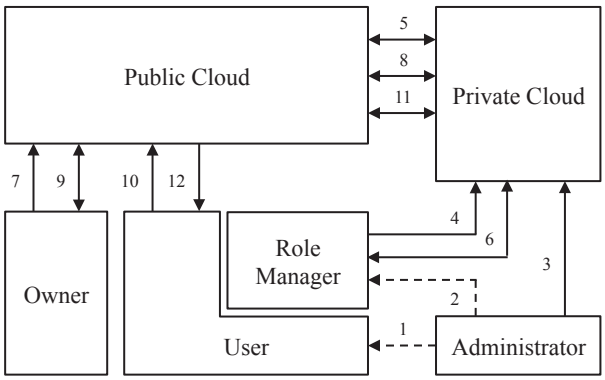


Fig. 3: RBE System Architecture

A. Architectural Components

We first consider the components of the system architecture shown in Fig. 3. The numbers shown in the figure refer to the system operations which will be described in Section V-B.

Public Cloud : Public cloud is a third party cloud provider which resides outside the infrastructure of the organisations, and organisations outsource their users' encrypted data to the public cloud. Since the public cloud is untrusted, data stored in the public cloud could be accessed by unauthorised parties, such as employees of the cloud provider and users from other organisations who are also using services from the same cloud. Therefore only public information and encrypted data will be stored in the public cloud.

An untrusted public cloud may deny a user's request for accessing stored data in the cloud or provide users with incorrect data. Such behaviours will result in the users not being able to access the data stored in cloud (cf denial of service attacks), but will not cause violation of RBAC policies. These behaviours can be detected, as a user can observe the failure immediately after s/he communicates with the public cloud. In this case, organisation may choose to change the cloud provider to a more reliable one, especially if the current provider is found to be malicious. The discussion of such denial of service attacks type behaviours are beyond the scope of this paper; hence in this paper, we will assume that the public cloud will faithfully execute the steps of the proposed RBE scheme and provide valid responses to users' requests.

Private Cloud : Private cloud is built on an internal data centre that is hosted and operated by a single organisation. The organisation only stores critical and confidential information in this private cloud. The amount of this information is relatively small comparing to the data stored in public cloud, so this cloud does not need to have the capacity to handle large volumes of data. The private cloud only provides interfaces to the administrator and role managers of the role-based system and to the public cloud. Users do not have direct access to the private cloud. This helps to reduce the attack surface of the private cloud. The purpose of using a private cloud is to ensure that correct and up-to-date information about the organisation's structure and user membership are used in the decision making. To achieve efficient user revocation, the private cloud is assumed to be honest-but-curious in order to use the proposed scheme in this architecture. That is, the cloud will faithfully execute the scheme and will not collaborate with revoked users.

User : Users are the parties who wish to acquire certain data from the public cloud. Each user is authenticated by the administrator of the role-based system; upon successful authentication, the user is given a secret key which is associated with the identity of the user. (In this paper, though we do not address the authentication, any one of suitable authentication schemes can be used for this purpose). Users are not involved in any process related to organisation structure updates, including user membership updates and changes in the role hierarchy. So they are not allowed to communicate directly with the private cloud.

Role Manager : A role manager is the party who manages the relationship between users and roles. Each role has its own role parameters which defines the user membership. These role parameters are stored in the private cloud. When updating the user membership of a role, the role manager needs to compute new role parameters and update them in the private cloud. None of users are affected by this operation, so role managers do not need to communicate with users, and they only need to interact with the private cloud. Before a user is included into a role, the role manager will need to authenticate the user in order to ensure that the user qualifies for the role. We do not consider the authentication mechanisms in this paper; we assume that such mechanisms exist and role managers will grant role membership only to appropriately qualified users in the system.

Administrator: The administrator is the certificate authority of the organisation. The administrator generates the system parameters and issues all the necessary credentials. In addition, the administrator manages the role hierarchy structure of the organisation. To put a role into the organisation's hierarchy structure, the administrator computes the parameters for that role. These parameters represent the position of the role in the role hierarchy, and are stored in the private cloud. When the role hierarchy changes, the administrator updates these parameters for the roles that have been changed in the private cloud.

Owner: Owners are the parties who possess the data and want to store the encrypted data in the public cloud for other users to access; owners specify who can access the data in terms of role-based policies. In the RBAC model, they are the parties who manage the relationship between permissions and roles. An owner can be a user within the organisation or an external party who wants to send data to users in the organisation. In this architecture, we consider an owner to be a logically separate component even though a user can be an owner and vice versa. Owners only interact with the public cloud, and no secret values are required for these interactions. They do not have to keep any parameters in the RBE scheme, and they need to obtain all the required parameters from the public cloud when they perform their encryption operations.

Secure Communications: We use ID-based signature (IBS) scheme in our system to certify the data communicated between the different parties. Using this technique the receiving party can verify the integrity of data content and authenticate the source of data. Shamir introduced the concept of identity based cryptography in 1984 [22], and he gave the first con-

struction of IBS. Since then, many different IBS schemes have been proposed; we have chosen the scheme proposed in [23] in our system implementation, as it has a similar key format to the RBE scheme. Please refer to Appendix B for details of the IBS scheme. With this IBS scheme, in our system, users use their secret decryption keys to sign the data. The private cloud is assigned an unique identity and regarded as a system user even though it is not granted membership to any role. Therefore the private cloud can sign data using its user secret key. We use $\mathcal{D}_R(M)$ to denote data M's signature, which is signed to the identity of R.

B. System Operations

Now we describe the system operations of our proposed architecture using the steps shown in Fig. 3. Assume the system uses a secure encryption scheme Enc to encrypt messages using the key generated in the *Encrypt* algorithm.

Extract: This operation is executed by the administrator to add a user or a role into the system. Step 1 represents the interaction to generate a decryption key for a user, and step 2 represents the interaction to generate a role secret for a role. The administrator computes the secret dk for the user or sk for the role, and sends the secret to the user or role via a secret channel; we denote a secret channel using a dotted line in this figure.

ManageRole: The operation of managing a role in the role hierarchy structure is also executed by the administrator. Steps 3 represents the interactions in the management of a role. The administrator decides the inheritance relationship of the role, and updates the position of a role in the role hierarchy structure. This is done in step 3 where the administrator computes and uploads the following tuple to the private cloud,

$$\langle \mathsf{ID}_R, A_R, B_R, \mathcal{P}_R \rangle$$

where ID_R is the identity of the role, A_R, B_R are computed as shown in the *ManageRole* algorithm, and $\mathcal{P}_R$ is the set of all the immediate roles that inherit permissions from the role ID_R.

Add User/Revoke User: These two operations are performed by role managers to update the user membership of roles, and the interactions for these operations are represented by steps 4 to 6. When adding or revoking users, a role manager sends the pair $\langle \mathsf{ID}_U, \tau \rangle$ (ID_U is the user identity and τ indicates the type of operation - add or revoke) to the private cloud in step 4. In step 5, the private cloud forwards this request to the public cloud, and the public cloud computes and returns Y_R to the private cloud. In step 6, the private cloud forwards the value Y_R to the role manager, and the role manager verifies Y_R, computes and uploads the following tuple to the private cloud,

$$\langle \mathsf{ID}_U, T_R, W_R, V_R, S_R \rangle$$

where ID_U is the identity of the user, and T_R, W_R, V_R, S_R are the values computed as in *AddUser*/*RevokeUser* algorithms.

Encryption: Steps 7 to 9 show the processes involved in encrypting a message. When an owner wants to encrypt data M to a role R, s/he retrieves the role public parameters from the public cloud as part of the encryption key, which is shown as step 7. Since these parameters are stored in the private cloud, the public cloud forwards the owner's request to the private cloud, and the private cloud passes back the following tuple to the public cloud in step 8,

$$\langle P = (\mathsf{ID}_R, A_R, B_R, t),\ \mathcal{D}_C(P) \rangle$$

where t is the current timestamp of the system, and $\mathcal{D}_C(P)$ denotes the signature of the message that is signed by the private cloud. In step 9, the public cloud simply forwards the tuple to the owner. Upon receiving the role public parameters, the owner checks if the timestamp t is up-to-date and verifies the attached signature to check its validity and whether it is issued by the private cloud. If the role public parameters are verified to be valid, then the owner computes and uploads the following ciphertext to the public cloud,

$$\langle C_1, C_2, C_3, Enc_k(M) \rangle$$

After receiving the ciphertext, the public cloud generates a unique index to identify the message, and stores this index-value pair in the cloud.

Decryption: Steps 10 and 12 show the processes involved in data decryption. When a user U wants to view the data M that has been previously encrypted and stored in the public cloud, the user first requests the ciphertext of M from the public cloud in step 10. Since the role parameters used in decryption are stored in the private cloud, the public cloud needs to request these parameters from the private cloud. The public cloud sends the following tuple to the private cloud,

$$\langle \mathsf{ID}_R, C_1, C_2, C_3 \rangle$$

where ID_R is the identity of the role that the user U belongs to and C_1, C_2, C_3 are parts of the ciphertext stored in the public cloud. The private cloud computes the value D, and returns the following tuple to the public cloud in step 11.

$$C = \langle\ P = (C_1, C_2, D, W_R, V_R, S_R, t), \mathcal{D}_C(P)\ \rangle$$

where t is the current timestamp of the system, and $\mathcal{D}_C(P)$ denotes the signature of the message signed by the private cloud.

In step 12, when the public cloud receives the above tuple, it computes and returns the following values to the user.

$$\langle\ C, g^{p_{i,\mathcal{M}}(s)}, g^{p_{k,\mathcal{N}}(s)}, Aux_1, Aux_2, Enc_K(M)\ \rangle$$

Note that the public cloud only calculates the values $h^{p_{i,\mathcal{M}}(s)}$, $h^{p_{k,\mathcal{N}}(s)}$ when the role hierarchy or user membership has changed. More specifically, $h^{p_{i,\mathcal{M}}(s)}$ is re-computed if the role parameter A_R has changed since the last decryption request from the same user, and $h^{p_{k,\mathcal{N}}(s)}$ is re-computed if the value V_R has changed since the last decryption request from the user. The user first checks if the timestamp t in C is up-to-date and verifies the attached signature to check its validity and whether it is issued by the private cloud. If the verification is successful, the user runs the *Decrypt* algorithm of the RBE scheme using the above values, and recovers the data M.

Recall in our architecture, ciphertext data is identified by unique indices in the cloud. When a user needs to access a certain data, the user will need to find its index first. To

TABLE II: Ciphertext Size (Bytes)

Plaintext Size	10 Roles	100 Roles	1000 Roles
1000	1432	1432	1432
10000	10432	10432	10432
100000	100432	100432	100432

allow users to be able to search for certain messages in the cloud, it is possible to integrate searchable encryption schemes in our system. A searchable encryption scheme allows us to make search queries to encrypted data without leaking information on both the queries and the data. Many searchable encryption schemes are proposed in public key systems, such as the constructions in [24]–[26]. Using one such searchable encryption schemes, a user can make a query to the public cloud about the data that s/he wants to view. The cloud returns a list of indices for the data that satisfy the query and which are accessible to the user; the cloud learns nothing about either the query or the content of the data. Since our aim is to build a system prototype of a secure cloud storage system, we have provided this functionality which we intend to use further in our future work on secure data searches and archiving.

A user in this architecture may want to authenticate the source of data, as the public cloud may return the wrong ciphertext data to mislead the user. It is easy to see that the data owner can employ an IBS scheme to sign the data and encrypt the signature together with the data. When the user decrypts the data, the user is able to verify the signature attached to the data to ensure it is created by a valid owner. If the owner is one of the users in the system, s/he can simply use the secret decryption key to sign the data using the IBS scheme described in Appendix B.

VI. Implementation

A. System Prototype

We have implemented the above architecture of the secure cloud data storage system. The system is implemented in Java. The interfaces of the cloud are exposed as JAX-WS [27] web services, and the web services are hosted in Apache Tomcat. The clouds use HyperSQL [28] database which can be easily replaced by other databases for server side data storage. The client side is written as Java Applet which can run in any Internet browser with Java support. To ensure that the client side gets the valid system public keys, these keys are embedded in the Java Applet, and the Applets are signed by the key generated by the trusted certificate authority when the Java Applet is compiled.

Our RBE scheme uses asymmetric bilinear groups, where the bilinear map takes inputs from two distinct isomorphic groups $\mathbb{G}_1$, $\mathbb{G}_2$. This allows a greater variety of pairings to be used in the implementation, especially certain families of non-supersingular elliptic curves [29]. In our implementation, we use a 163-bit MNT curves [30] with the embedding degree of 6. In practice, the security of a 160-bit elliptic curve is roughly equivalent to 1024-bit RSA [31]. We use SHA-1 to map the identities to points on the elliptic curve as the output size of SHA-1 is 160-bit, which is of similar length as the input of the pairing.

We use ISAAC [32] as the symmetric encryption algorithm Enc; the reason for this choice is that a stream cipher can work with the MTOM [33] feature of the web service to perform encryption and decryption while transferring the data. Moreover, ISAAC takes keys from 8-bit to 8288-bit length, so the output of the pairing can be directly used as the symmetric encryption key without being transformed. We consider ISAAC as a secure symmetric encryption algorithm as the attack complexity is 4.67×10^{1240} [34] for the best known attack to ISAAC [35].

We use jPBC [36] and PBC [37] as our pairing-based crypto library (jPBC wraps the PBC library to generate a MNT curve), and Bouncy Castle crypto library [38] for SHA-1 and ISAAC.

B. Experimental Evaluation

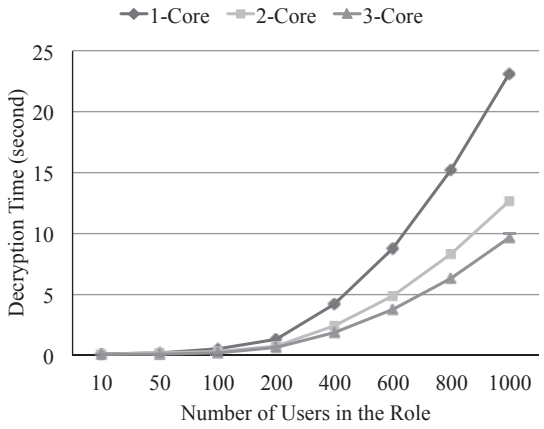
We have performed our experiments on a cluster of three machines, each with a quad-core Intel Q6600 2.40 GHz processor, 4 GB of RAM, two 7200 RPM hard disks, that were connected by gigabit switched Ethernet.

Let us first consider the size of the ciphertext. From the description of the RBE scheme, we see that the ciphertexts do not contain user related information, but are computed using the parameters containing the identities of all the ancestor roles of the target role, we compare the size of the ciphertext when the target role has 10, 100, 1000 ancestor roles respectively.

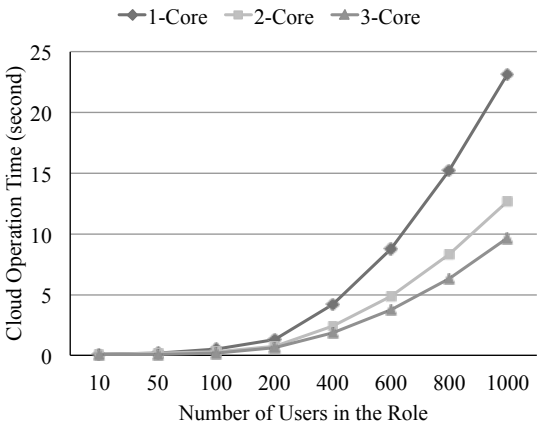
Table II shows the ciphertext sizes when the sizes of plaintext are 1000 Bytes, 10000 Bytes, 100000 Bytes respectively. First we note that the differences in size between the plaintext and ciphertext are constant. Secondly, the ciphertext size remains the same when the number of ancestor roles changes. We conclude that the ciphertext size is linearly proportional to the size of the plaintext regardless of the number of roles and users who can decrypt the ciphertext.

The size of the decryption key is another important factor in cloud storage system. The decryption key needs to be portable as users may use the storage service from different clients. Our experimental results show that the size of the decryption key is 48 bytes, which is convenient for the users. A non-constant size decryption key will usually make it difficult for the users to decide the memory requirements that are needed on the client devices to store the keys. Our system does not have this problem.

Encryption and decryption are the most frequently used operations in the system. Since we have split decryption algorithm to run it in both client and the cloud, we first measured the time taken at the cloud for performing decryption. The time for cloud decryption is measured from the time the public cloud receives the computed role parameters from the private cloud, to the time the cloud starts sending the ciphertext to the user. We have split the computation task of the cloud decryption into multiple threads. This approach helps to reduce the decryption time in the cloud, as the cloud can have multiple processor cores on demand running these multiple threads. In our experiments, we have simulated increasing number of processor cores by increasing the number of running threads. Since we use one thread as the master thread in the computation task, the maximum cores that we can simulate on our quad-core server is 3.

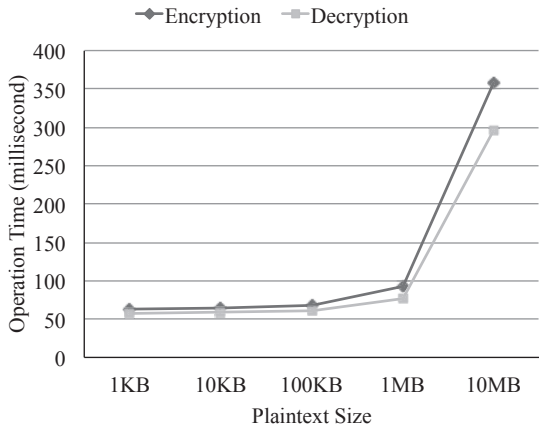


(a) Public Cloud Decryption Time

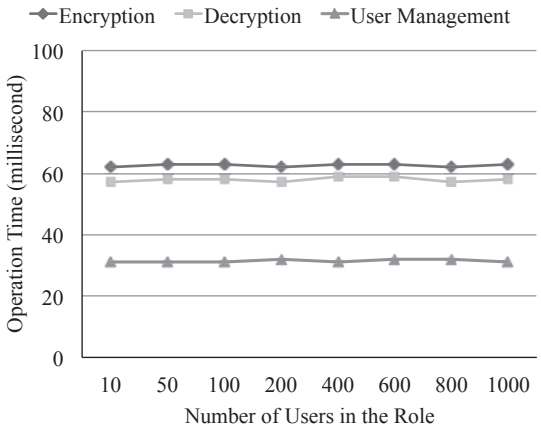


(b) Public Cloud Operation Time in User Management

Fig. 4: Public Cloud Operation Time



(a) Different Plaintext Size



(b) Different Number of Users

Fig. 5: Client Operation Time

Fig. 4(a) shows the time that the cloud server has spent in executing the decryption algorithm on a ciphertext of a 1KB file when different number of users are in the role to which the user performing the decryption belongs. In this case, 4 roles have been created as the ancestor roles of this role. Increasing the number of ancestor roles of the role that the user belongs to also increases the decryption time; increasing the number of ancestor roles has the same impact on the cloud decryption time as increasing the number of users does. However it is important to note that the number of roles is usually much smaller than the number of users involved in each role.

We note that the cloud decryption time is the cloud's *response time* to users' decryption requests. This is the time that users need to wait after they have initiated decryption requests to the cloud. To have good user experience in using this cloud storage system, this time needs to be small. We have conducted a series of experiments with 1, 2 and 3 cores to perform the decryption. From the results, we note that increasing the number of processor cores participating in the decryption shortens the *response time* of the cloud. When deployed in a real cloud environment which has a large number of virtual processor cores, we believe that the cloud response time can be controlled to a suitable range that is acceptable to the users. Recall that the cloud caches the result of the cloud decryption. So it does not have to compute for every decryption request, which reduces the average cloud decryption time even further.

In our system, role managers also outsource part of the computations to the cloud which are concerned with user management. Fig. 4(b) shows the time that the cloud has spent in collaborating with role managers on this computation; in this experiment, we have created 4 roles as the ancestors of this role. Similar to the cloud decryption, the time for this computation can be reduced by increasing the number of processor cores.

Next we look at the client operation time. Fig. 5(a) shows the time for encrypting and decrypting files of different sizes on the client side. In this experiment, we created 5 roles and 10

users in each role. In our measurements, the encryption time was measured from the time when an owner clicks on the upload button in the Java Applet after choosing the file to be encrypted, to the time when the file upload has been completed and the owner receives the cloud's response indicating that the transaction successful. The decryption time was measured from the time when a user starts receiving the ciphertext from the cloud till the time the plaintext is saved to a file on the local disk drive.

Since we are using the stream cipher ISAAC in our implementation, the encryption/decryption of data can happen while the data is being transferred over the network. In the encryption/decryption algorithm of the RBE scheme, a key is computed and used in the symmetric encryption scheme. As seen from Fig. 5(a), when the size of the plaintext is smaller than 100KB, the time for the symmetric encryption/decryption including data transfer is trivial compared to the time for the symmetric key generation by the RBE scheme. Hence the time in our measurement remains the same. When the size of the plaintext exceeds 1MB, the time for the symmetric encryption/decryption increases to a similar order as that of the time for symmetric key generation. Hence we notice the increase in the time with the growth of the size of the plaintext.

By outsourcing these heavy computations to the cloud, the operations' time of clients is reduced dramatically. Fig. 5(b) shows the time that we measured for the encryption of 1KB data by the owner, decryption of 1KB data by the users, and the user management of role managers (when a role contains different number of users). Once again there are 4 ancestor roles for this role. The results show that the time for these operations is somewhat constant regardless of the number of users and roles involved in the computation. Hence it would be possible to perform these operations on mobile devices with less computational power.

VII. Conclusion

In this paper, first we proposed a new RBE scheme that achieves efficient user revocation. Then we presented a RBAC based cloud storage architecture which allows an organisation to store data securely in a public cloud, while maintaining the sensitive information related to the organisation's structure in a private cloud. Then we have developed a secure cloud storage system architecture and have shown that the system has several superior characteristics such as constant size ciphertext and decryption key. From our experiments, we observe that both encryption and decryption computations are efficient on the client side, and decryption time at the cloud can be reduced by having multiple processors, which is common in a cloud environment. We believe that the proposed system has the potential to be useful in commercial situations as it captures practical access policies based on roles in a flexible manner and provides secure data storage in the cloud enforcing these access policies.

References

[1] F. R. Institute, "Personal data in the cloud: A global survey of consumer attitudes," http://www.fujitsu.com/downloads/SOL/fai/reports/fujitsu_personal-data-in-the-cloud.pdf, 2010.

[2] KPMG, "From hype to future: Kpmg's 2010 cloud computing survey," http://www.kpmg.com/NL/nl/IssuesAndInsights/ArticlesPublications/Documents/PDF/IT%20Performance/From_Hype_to_Future.pdf, 2010.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[4] Avanade, "Global survey: Has cloud computing matured?" http://www.avanade.com/Documents/Research%20and%20Insights/Global_Survey_Slide_Graphics_Has_Cloud_Matured.pdf, 2011.

[5] C. Delerablée, "Identity-based broadcast encryption with constant size ciphertexts and private keys," in *ASIACRYPT*, ser. Lecture Notes in Computer Science, vol. 4833. Springer, 2007, pp. 200–215.

[6] L. Zhou, V. Varadharajan, and M. Hitchens, "Enforcing role-based access control for secure data storage in the cloud," *The Computer Journal*, vol. 54, no. 13, pp. 1675–1687, October 2011.

[7] Y. Zhu, H. Hu, G.-J. Ahn, H. Wang, and S.-B. Wang, "Provably secure role-based encryption with revocation mechanism," *J. Comput. Sci. Technol.*, vol. 26, no. 4, pp. 697–710, 2011.

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

[9] M. J. Atallah, K. B. Frikken, and M. Blanton, "Dynamic and efficient key management for access hierarchies," in *ACM Conference on Computer and Communications Security*. ACM, November 7-11 2005, pp. 190–202.

[10] H. R. Hassen, A. Bouabdallah, H. Bettahar, and Y. Challal, "Key management for content access control in a hierarchy," *Computer Networks*, vol. 51, no. 11, pp. 3197–3219, 2007.

[11] S. D. C. D. Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Over-encryption: Management of access control evolution on outsourced data," in *VLDB*. ACM, September 23-27 2007, pp. 123–134.

[12] C. Blundo, S. Cimato, S. D. C. di Vimercati, A. D. Santis, S. Foresti, S. Paraboschi, and P. Samarati, "Efficient key management for enforcing access control in outsourced scenarios," in *SEC*, ser. IFIP, vol. 297. Springer, May 18-20 2009, pp. 364–375.

[13] P. Samarati and S. D. C. di Vimercati, "Data protection in outsourcing scenarios: issues and directions," in *ASIACCS*. ACM, April 13-16 2010, pp. 1–14.

[14] C. Gentry and A. Silverberg, "Hierarchical id-based cryptography," in *ASIACRYPT*, ser. Lecture Notes in Computer Science, vol. 2501. Springer, 2002, pp. 548–566.

[15] D. Boneh, X. Boyen, and E.-J. Goh, "Hierarchical identity based encryption with constant size ciphertext," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, vol. 3494. Springer, May 22-26 2005, pp. 440–456.

[16] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *ACM Conference on Computer and Communications Security*. ACM, October 30 - November 3 2006, pp. 89–98.

[17] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *EUROCRYPT*, 2005, pp. 457–473.

[18] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *INFOCOM*. IEEE, 15-19 March 2010, pp. 534–542.

[19] Y. Zhu, D. Ma, C. Hu, and D. 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*, ser. Cloud Computing '13. ACM, 2013, pp. 33–40.

[20] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh, "Sirius: Securing remote untrusted storage," in *NDSS*. The Internet Society, 2003.

[21] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," in *NDSS*. The Internet Society, 2005.

[22] A. Shamir, "Identity-based cryptosystems and signature schemes," in *CRYPTO*, ser. Lecture Notes in Computer Science, vol. 196. Springer, 1984, pp. 47–53.

[23] P. S. L. M. Barreto, B. Libert, N. McCullagh, and J.-J. Quisquater, "Efficient and provably-secure identity-based signatures and signcryption from bilinear maps," in *ASIACRYPT*, ser. Lecture Notes in Computer Science, vol. 3788. Springer, December 4-8 2005, pp. 515–532.

[24] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, vol. 3027. Springer, 2004, pp. 506–522.

[25] P. Golle, J. Staddon, and B. R. Waters, "Secure conjunctive keyword search over encrypted data," in *ACNS*, ser. Lecture Notes in Computer Science, vol. 3089. Springer, June 8-11 2004, pp. 31–45.
[26] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *TCC*, ser. Lecture Notes in Computer Science, vol. 4392. Springer, February 21-24 2007, pp. 535–554.
[27] JAX-WS, "Jax-ws reference implementation," http://jax-ws.java.net/.
[28] HyperSQL, "Hypersql database," http://hsqldb.org/.
[29] J. H. Silverman, *The Arithmetic of Elliptic Curves*, 2nd ed., ser. Graduate Texts in Mathematics. Springer, 2009, vol. 106.
[30] A. Miyaji, M. Nakabayashi, and S. Takano, "New explicit conditions of elliptic curve traces for fr-reduction," *IEICE Transactions on Fundamentals*, vol. E84-A, no. 5, pp. 1234–1243, 2001.
[31] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "Recommendation for key management - part 1: General (revision 3)," NIST, Tech. Rep., May 2011.
[32] R. J. J. Jr., "Isaac," in *FSE*, ser. Lecture Notes in Computer Science, vol. 1039. Springer, February 21-23 1996, pp. 41–49.
[33] W3C, "Soap message transmission optimization mechanism," http://www.w3.org/TR/soap12-mtom/.
[34] M. Pudovkina, "A known plaintext attack on the isaac keystream generator," Cryptology ePrint Archive, Report 2001/049, 2001.
[35] J.-P. Aumasson, "On the pseudo-random generator isaac," Cryptology ePrint Archive, Report 2006/438, 2006.
[36] A. D. Caro and V. Iovino, "Java pairing based cryptography library," http://libeccio.dia.unisa.it/projects/jpbc/.
[37] B. Lynn, "Pairing-based cryptography library," http://crypto.stanford.edu/pbc/.
[38] BouncyCastle, "Bouncy castle cryptography library," http://www.bouncycastle.org/.
[39] R. Canetti, S. Halevi, and J. Katz, "Chosen-ciphertext security from identity-based encryption," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, vol. 3027. Springer, 2004, pp. 207–222.
[40] D. Boneh and J. Katz, "Improved efficiency for cca-secure cryptosystems built using identity-based encryption," in *CT-RSA*, ser. Lecture Notes in Computer Science, vol. 3376. Springer, February 14-18 2005, pp. 87–103.

APPENDIX A
SECURITY PROOF FOR THE RBE SCHEME IN SECTION IV

A. Security Properties

First we define the security properties of our RBE scheme. We build the security of the scheme on the standard notion of selective-ID security. In our RBE scheme, users who have never been granted the membership of a role as well as the users who have been granted the membership but being revoked later should not have the ability to decrypt the data that is encrypted to the role. Since the revoked users have the access to the secrets of the role before they are revoked, we assume that the adversary $\mathcal{A}$ has more power when making the query to the challenger; the adversary can make queries on the identities in the target set before the **Challenge** stage.

We say that our RBE scheme is secure against chosen ciphertext attack (CCA) if no polynomially bounded adversary $\mathcal{A}$ has a non-negligible advantage against the challenger in the following game:

Init: The adversary $\mathcal{A}$ first chooses a set of identities $U = \{\mathsf{ID}_{U_1}, \cdots, \mathsf{ID}_{U_n}\}$ which $\mathcal{A}$ will query to the challenger.

Setup: The challenger takes as input a security parameter and runs the *Setup* algorithm. It outputs the system public key pk to the $\mathcal{A}$, and keeps the master key mk as secret. Then the challenger creates a set of roles with the public identities $R = \{\mathsf{ID}_{R_1}, \cdots, \mathsf{ID}_{R_m}\}$ where $R \cap U = \varnothing$ by running the *Extract* algorithm to generate the role secrets and running the *ManageRole* algorithm to organise them in a hierarchical structure.

Phase 1: The adversary $\mathcal{A}$ can adapt its queries depending upon the results of the previous queries. The adversary issues queries $q_1, \ldots, q_k$ where each query is one of the following type:

- *Extract query:* $\mathcal{A}$ submits an identity $\mathsf{ID} \notin R$ to the challenger. The challenger executes the *Extract* algorithm on the identity ID and returns the generated key to $\mathcal{A}$.
- *AddUser query:* $\mathcal{A}$ submits two identities ID_{U_i} and ID_{R_j} to the challenger. Here we allow ID_{U_i} and ID_{R_j} to be selected from the set U and R respectively. If ID_{U_i} has already been included in the role ID_{R_j}, the challenger returns the key dk_i and the public parameters pub_j to $\mathcal{A}$. Otherwise, the challenger executes the *AddUser* algorithm to output dk_i, pub_j and returns them to $\mathcal{A}$.
- *Decrypt query:* $\mathcal{A}$ submits a tuple $\langle C, \mathsf{ID}_{R_j}, \mathsf{ID}_{U_i}\rangle$ to the challenger. The challenger executes the *Decrypt* algorithm and returns the result to $\mathcal{A}$.

Challenge: When the adversary $\mathcal{A}$ decides that Phase 1 is completed, it outputs role identity ID_R on which it wishes to be challenged. Similarly, we allow ID_R to be one of the identities that appears in the query in Phase 1. However, the challenger checks if there is any identity that has been queried together with the identity ID_R in the *AddUser* query. If there are some, the challenger executes *RevokeUser* algorithm to exclude these identities from the role and updates pub_R. When no identity queried in **Phase 1** belongs to the role R, the challenger runs the *Encrypt* algorithm and outputs (C^*, K) where $K \in \mathcal{K}$. Next, the challenger picks a random bit $b \in \{0, 1\}$ and sets $K_b = K$, and then it picks a random $K_{1-b} \in \mathcal{K}$, and returns (C^*, K_0, K_1) to $\mathcal{A}$.

Phase 2: The adversary $\mathcal{A}$ again adapts its queries and issues $q_{k+1}, \ldots, q_{q_T}$ similar to **Phase 1** with the following restrictions: $\mathcal{A}$ cannot make *Extract* or *AddUser* queries on ID_R, and $\mathcal{A}$ cannot make *Decrypt* queries on C^*.

Guess: The adversary $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$, and wins the game if $b = b'$.

We denote the probability that $\mathcal{A}$ wins the game as $\mathsf{Adv}^{\mathcal{RBE}}$ and we have the following definition.

Definition 1: We say that a role-based encryption scheme is $(\epsilon, n, q_E, q_A, q_D)$ CCA secure if for all polynomially bounded adversary $\mathcal{A}$ who has made a total of q_E extraction queries, q_A add-user queries, and q_D decryption queries, we have $|\mathsf{Adv}^{\mathcal{RBE}} - \frac{1}{2}| < \epsilon$

Next we give the definition of the chosen plaintext attack (CPA) secure of a RBE scheme by not issuing the decrypt queries in the above game.

Definition 2: We say that a role-based encryption scheme is CPA secure if it is $(\epsilon, n, q_E, q_A, 0)$ CCA secure.

B. Security Assumptions

A General Diffie-Hellman Exponent Problem assumption ((P, Q, f)-GDHE) has been introduced and extended to the General Decisional Diffie-Hellman Exponent assumption (GDDHE) in [15], which has been proved to have generic

security with GDHE problem. The security of our scheme is based on this assumption. First, let us review the GDDHE problem.

Let p be an integer prime and let s, n be positive integers. Let $P, Q \in \mathbb{F}_p[X_1, \ldots, X_n]^s$ be two s-tuples of n-variate polynomials over $\mathbb{F}_p$ and let $f \in \mathbb{F}_p[X_1, \ldots, X_n]$. Let $P = (p_1, p_2, \ldots, p_s)$ and $Q = (q_1, q_2, \ldots, q_s)$, and the first components of P, Q satisfy $p_1 = q_1 = 1$. The *(P, Q, f)*-GDDHE Problem is defined as follows,

Definition 3: **(P, Q, f)-GDDHE Problem** Let $(p, \mathbb{G}, \mathbb{G}_T, \hat{e})$ be a bilinear map group system and let g be the generator of $\mathbb{G}$, and $v = \hat{e}(g, g)$. Given the tuple

$$(g^{P(x_1,\ldots,x_n)}, v^{Q(x_1,\ldots,x_n)}) \in \mathbb{G}^s \times \mathbb{G}_T^s$$

and $T \in \mathbb{G}_T$, decides whether T is equal to $v^{f(x_1,\ldots,x_n)}$.

C. Security Proof

Chosen Plaintext Security. In this subsection, we first prove CPA security of our RBE scheme based on GDDHE assumption.

Theorem 1: The proposed RBE scheme is chosen plaintext secure under the $GDDHE$ assumption.

To prove Theorem 1, we start by defining a specific GDDHE problem where it clearly shows that $f = zh(s)$ is independent from P and Q. In the following definition, s, k, z, t are random elements chosen from $\mathbb{Z}_p^*$ as described in our RBE scheme.

Definition 4: **(f, g, h, F)-GDDHE Problem** Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e})$ be a bilinear map group system and let g_0 and h_0 be the generator of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and $v = \hat{e}(g, h)$. Given $K \in \mathbb{G}_T$ and

$$g_0, g_0^s, \ldots, g_0^{s^{m-1}},\ g_0^{g(s)},\ g_0^{kg(s)},\ g_0^{tg(s)},\ g_0^{ktg(s)},\ g_0^{h(s)}$$

$$h_0, h_0^s, \ldots, h_0^{s^{n-1}}, h_0^{sf(s)}, h_0^{s^2 f(s)}, h_0^{zsf(s)}$$

$$h_0^{h(s)}, h_0^{kh(s)}, h_0^{zh(s)}, h_0^{kzh(s)}$$

in deciding whether K is equal to $\hat{e}(g_0, h_0)^{zh(s)}$ or some random element of $\mathbb{G}_T$.

Given an attacker $\mathcal{A}$ who wins the following game with probability $\mathsf{Adv}_{\mathcal{A}}^{\mathcal{RBE}}$, we construct another attacker $\mathcal{B}$ that solves the $GDDHE$ problem. In our scheme, we assume that cloud is trusted, and will not collude with malicious adversaries.

Let q be the maximum number of identities of users and roles that the adversary can query. Let $\mathcal{U} = \{\mathsf{ID}_{U_1}, \ldots, \mathsf{ID}_{U_n}\}$ and $\mathcal{R} = \{\mathsf{ID}_{R_1}, \ldots, \mathsf{ID}_{R_m}\}$ be the sets of users and role identities respectively that the adversary will issue queries on. $\mathcal{B}$ will be given a (f, g, F)-GDDHE instance defined in Definition 4. Next we define the following polynomials:

- $g(x) = \prod_{i=1}^{m}(x + H_1(\mathsf{ID}_{R_i}))$
- $f(x) = \prod_{i=1}^{n}(x + H_1(\mathsf{ID}_{U_i}))$
- $h(x) = g(x) \cdot f(x)$
- $g_i(x) = \frac{g(x)}{x + H_1(\mathsf{ID}^*_{R_i})}$, where $i \in [1, m]$
- $f_i(x) = \frac{f(x)}{x + H_1(\mathsf{ID}^*_{U_i})}$, where $i \in [1, n]$

Init: The adversary $\mathcal{A}$ first chooses a set of identities $U = \{\mathsf{ID}_{U_1}, \cdots, \mathsf{ID}_{U_n}\}$ which $\mathcal{A}$ will attack (with $n \leq q$).

Setup: $\mathcal{B}$ will set the following values to generate the system parameters

$$g = g_0^{g(s)},\ w = h_0^{s \cdot f(s)},\ w^s = h_0^{s^2 \cdot f(s)}$$

$$v = \hat{e}(g, h) = \hat{e}(g_0, h_0)^{h(s)}$$

This implies that $h = h_0^{f(s)}$. Then $\mathcal{B}$ defines the public key as $\mathsf{pk} = (w, v, g^k, g, g^s, \cdots, g^{s^q})$, and creates a role ID_R by outputting the role parameters

$$A = h_0^{h(s)} = h^{g(s)} = h^{\prod_{i=1}^{m}(s + H_1(\mathsf{ID}^*_{R_i}))},$$

$$B = h_0^{kh(s)} = A^k$$

Phase 1: The adversary $\mathcal{A}$ once again adapts its queries and issues queries $q_1, \ldots, q_{q_k}$,

- *Extract query:* If $\mathcal{A}$ has not issued the query on ID_{U_i}, $\mathcal{B}$ computes the user decryption key as

$$\mathsf{dk}_{U_i} = h^{f_i(s)} = h_0^{\frac{1}{s + H_1(\mathsf{ID}^*_{U_i})}}$$

- *AddUser query:* If $\mathcal{A}$ has issued the query on ID_{U_i} and the role created in **Setup**, $\mathcal{B}$ then updates the role public parameters for the role to grant the role membership to the user. When n users are the member of the role, the role parameters $\mathcal{B}$ outputs will be

$$W = h_0^{-r' s f(s)} = w^{-r'},$$

$$V = g_0^{r' h(s)} = g^{r' f(s)} = g^{r' \cdot \prod_{i=1}^{n}(s + H_1(\mathsf{ID}^*_{U_i}))}$$

$$S = H_2(\hat{e}(g_0, h_0)^{r' h(s)}) \cdot g_0^{g_i(s)} \cdot g_0^{t' kg(s)}$$

where r' and t' are the random number chosen by $\mathcal{B}$.

Challenge: Once $\mathcal{A}$ decides that Phase 1 is over, $\mathcal{B}$ first revokes all the members of the role ID_R. $\mathcal{B}$ outputs the following role parameters when all the users have been revoked from the role.

$$W = h_0^{-rsf(s)} = w^{-r},\ V = g_0^{rg(s)} = g^r$$

$$S = H_2(\hat{e}(g_0, h_0)^{rh(s)}) \cdot g_0^{g_i(s)} \cdot g_0^{ktg(s)}$$

where r is the random number chosen by $\mathcal{B}$.
Then $\mathcal{B}$ simulates *Encrypt* algorithm by constructing the ciphertext C^* for the message M_b for a random $b \in \{0, 1\}$ as,

$$C_1 = h_0^{-zsf(s)} = w^{-z},\ C_2 = h_0^{zh(s)} = A^z$$

$$C_3 = h_0^{kzh(s)},\ D = \hat{e}(g_0, h_0)^{-tg(s)kzh(s)},\ K = K'$$

Note that if $K' = \hat{e}(g_0, h_0)^{zh(s)}$, then we have $K = v^z$

Phase 2: The adversary $\mathcal{A}$ again adapts its queries and issues $q_{q_k+1}, \ldots, q_{q_T}$ similar to **Phase 1** with the following restrictions: $\mathcal{A}$ cannot make *Extract* or *AddUser* queries on ID_R, and $\mathcal{A}$ cannot make *Decrypt* queries on C^*.

Guess: The adversary $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$, and wins the game if $b = b'$.

Then we have

$$\begin{aligned}\mathsf{Adv}^{gddhe} &= \frac{1}{2}(Pr[b=b'|real]+Pr[b=b'|rand])-\frac{1}{2}\\ &= \frac{1}{2}(\frac{1}{2}+\mathsf{Adv}_{\mathcal{A}}^{\mathcal{RBE}})+\frac{1}{2}\cdot\frac{1}{2}-\frac{1}{2}\\ &= \frac{1}{2}\cdot\mathsf{Adv}_{\mathcal{A}}^{\mathcal{RBE}}\end{aligned}$$

Chosen Ciphertext Security. In above, we have proved that the proposed RBE scheme is chosen plaintext secure. There exist generic conversion methods [39], [40] to convert CPA-secure scheme to CCA-secure scheme. Our RBE scheme can be extended to a CCA-secure scheme using similar approach, such as using strongly unforgeable signatures.

APPENDIX B
ID-BASED SIGNATURE SCHEME IN [23]

In this section, we describe the IBS scheme proposed in [23]. However, in order to make the scheme cater for the same key format as the RBE scheme in our system, we describe a modified scheme by switching the usage of the two input groups of the pairings as follows, *Setup:* Generate three groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, and an bilinear map $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Randomly choose two generators $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$, a secret value $s \leftarrow \mathbb{Z}_p^*$ and two hash functions $H_1: \{0,1\}^* \rightarrow \mathbb{Z}_p^*$, $H_2: \{0,1\}^* \times \mathbb{G}_T \rightarrow \mathbb{Z}_p^*$ The master secret key mk and system public key pk are defined as

$$\mathsf{mk} = \{s\}, \quad \mathsf{pk} = \{w, v, g\} \text{ where } w = g^s, v = \hat{e}(g,h)$$

KeyGen: For an identity ID, the private key is generated as

$$\mathsf{sk}_{\mathsf{ID}} = h^{\frac{1}{s+H_1(\mathsf{ID})}}$$

Sign: To sign a message $M \in \{0,1\}^*$, the signer picks a random value $x \leftarrow \mathbb{Z}_p^*$ and computes $r = v^x$. The signature is output as

$$\sigma = (H, S) \text{ where } H = H_2(M, r), S = \mathsf{sk}_{\mathsf{ID}}^{(x+H)}$$

Verify: a signature $\sigma = (H, S)$ on a message M is accepted if the following equation holds

$$H = H_2(M, \hat{e}(g^{H_1(\mathsf{ID})} \cdot w, S) \cdot v^{-H})$$