Cryptographic Role-based Access Control for Secure Data Storage in Cloud Systems

By

Lan Zhou

A thesis submitted to Macquarie University
for the degree of
Doctor of Philosophy
Department of Computing
July 2014

MACQUARIE
UNIVERSITY
FACULTY OF SCIENCE

Typeset in LaTeX 2_ε.

Except where acknowledged in the customary manner, the material presented in this thesis is, to the best of my knowledge, original and has not been submitted in whole or part for a degree in any university.

Lan Zhou

Acknowledgements

First of all, I would like to express my deepest appreciation to my supervisor Professor Vijay Varadharajan, for his guidance and support to my research work. His excellent advice has inspired me to find the right direction when I lost my path. His supports for my travels have provided me invaluable opportunities to visit other academies and exchange ideas with other researchers.

I would also like to thank my co-supervisor Associate Professor Michael Hitchens for all the discussions and the comments on my research work, and his kind help with my paper writing.

In addition, I would like to thank Mrs Andrina Brennan, for her great help with all kinds of administrative works, which has made my work much easier in the past few years.

Finally, I am extremely grateful to my parents and my wife for their support and patient during the course of this work.

Abstract

With the rapid increase in the amount of digital information that needs to be stored, there has been a growing trend in recent times to store data in the cloud because of the benefits it provides such as on-demand access and scalability. Cloud data storage raises the important security issue of how to control and prevent unauthorised access to data stored in the cloud. Access control is required to specify who can create the access policies as well as to determine the access mechanisms needed to control access to the stored data. One well-known access control model is the role-based access control (RBAC) model, which provides flexible controls and security management by having two mappings, users to roles and roles to privileges on data objects.

We propose role-based encryption (RBE) schemes, which integrate cryptographic techniques with RBAC models, to enforce RBAC policies for data stored in the cloud. Using RBE schemes, the owners of data can encrypt it in such a way that only users with appropriate roles as specified by the access policy can decrypt and view the content of the data. We then describe the design of a secure RBE-based hybrid cloud storage architecture as well as a practical implementation of the proposed RBE-based architecture and its performance results.

In large-scale RBAC systems, decentralising the administration tasks is an important issue as it is usually impractical to centralise the task of managing these users and permissions, and their relationships with roles. We propose a cryptographic administrative model AdC-RBAC to manage and enforce role-based access policies for the RBE schemes in large-scale cloud systems. The AdC-RBAC model uses cryptographic techniques to ensure that administrative tasks such as user, permission, and role management are performed only by authorised administrative roles.

We have also demonstrated the suitability of the proposed RBE schemes and the developed architecture in two practical applications, one involving secure data storage in the cloud for a banking organisation and the other concerned with secure storage of patient-centric health records in the cloud.

The issue of trust is critical in cloud storage systems and it is necessary to address explicitly trust issues in the enforcement of access policies in cloud data storage as often implicit assumptions are made that the various system entities behave properly. We have developed trust models that can help to reason about the behaviour of users, data owners and role managers, taking into account role hierarchy and permission inheritance. We then describe the design of trust-enhanced secure cloud data storage systems integrating trust models and RBE schemes, which reduce risk and improve the quality of cloud storage services.

List of Publications

- Lan Zhou, Vijay Varadharajan, and Michael Hitchens. Enforcing role-based access control for secure data storage in the cloud. *The Computer Journal*, 54(13):1675–1687, October 2011. The Computer Journal Wilkes Award, British Computer Society.

- Lan Zhou, Vijay Varadharajan, and Michael Hitchens. A flexible cryptographic approach for secure data storage in the cloud using role-based access control. *International Journal of Cloud Computing*, 1(2/3):201–220, 2012.

- Lan Zhou, Vijay Varadharajan, and Michael Hitchens. Trusted administration of large-scale cryptographic role-based access control systems. In *The 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 714–721. IEEE Computer Society, June 25-27, 2012, Liverpool, UK.

- Lan Zhou, Vijay Varadharajan, and Michael Hitchens. Integrating trust with cryptographic role-based access control for secure cloud data storage. In *The 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 560–569. IEEE Computer Society, July 16-18, 2013, Melbourne, Australia.

- Lan Zhou, Vijay Varadharajan, and Michael Hitchens. Trust-based secure cloud data storage with cryptographic role-based access control. In *Proceedings of the*

10th International Conference on Security and Cryptography (SECRYPT), pages 62–73. SciTePress, July 29-31, 2013, Reykjavik, Iceland.

- Lan Zhou, Vijay Varadharajan, and Michael Hitchens. Cryptographic Role-Based Access Control for Secure Cloud Data Storage Systems, part III, pages 313–344. *Security, Privacy and Trust in Cloud Systems*, Springer, 2014.

- Lan Zhou, Vijay Varadharajan, and Michael Hitchens. Achieving Secure Role-based Access Control on Encrypted Data in Cloud Storage. *IEEE Transactions on Information Forensics and Security*, 8(12):1947–1960, 2013.

- Lan Zhou, Vijay Varadharajan, and Michael Hitchens. Secure Administration of Cryptographic Role-based Access Control for Large-Scale Cloud Storage Systems, accepted by special issue in *Journal of Computer and System Sciences* on Theory and Applications in Parallel and Distributed Computing.

- Lan Zhou, Vijay Varadharajan, and Michael Hitchens. Generic Constructions for Role-based Encryption. Technical Report, 2013, submitted to *International Journal of Information Security*, Springer.

Contents

List of Figures

List of Tables

1

Introduction

Cloud computing has attracted much attention in recent times because of its ability to deliver resources such as computing and storage to users on demand in a cost effective manner. Due to the continuous growth in the amount of digital information that needs to be stored, there is a clear incentive for service providers to explore outsourcing of users' data to the cloud. Potentially there could be several benefits to storing data in the cloud. The storage capacity of the cloud is almost unlimited, and users only need to pay for the storage space that they use for their actual needs. Outsourcing data to the cloud can also help to save the costs and efforts in storage maintenance tasks, such as data backup and replication, disaster recovery and hardware maintenance. Furthermore, cloud storage can provide a flexible and convenient way for users to access their data from anywhere on any device.

There are different types of infrastructures associated with a cloud [4, 100]. A public cloud is a cloud that 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. A community cloud is a cloud shared between several organisations from a specific community, and a hybrid cloud is a composition of two or more of the above described types of clouds. Since the public cloud is the most widely used infrastructure for data storage, in this thesis, when we use the word cloud, we are referring to the public cloud unless explicitly specified.

Though cloud storage has many benefits, it raises several important security issues. Since data in the cloud is stored in one or more data centres which are 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. As the cloud is an open platform, and can be subjected to malicious attacks from both insiders and outsiders, the need to protect the privacy and security of the data in the cloud becomes a critical issue.

An important security aspect when data is stored in the cloud is who is able to access and view it. In order to allow users to control the access to their data stored in a cloud, suitable access control policies and mechanisms are required. The access policies must restrict data access to only those intended by the data owners (users who own the data). 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. However, a data owner may not wish the cloud provider itself to view and access the data that is being stored in the cloud. Typically when we refer to a cloud provider, it can include several or many employees of the cloud provider organisation. The greater the sensitivity of the data stored in the cloud, the more stringent are the data owners' security requirements on the access to data.

1.1 Motivation

One common approach to protect the security and privacy of data stored in a cloud is using access controls. In traditional access control systems, enforcement of access policies is carried out by trusted parties, which are usually the service providers themselves. 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. One potential solution to these problems is for the users to employ cryptographic techniques to encrypt their data before storing it in the cloud. Data owners can encrypt the data in such a way that only users who are allowed to access the data as specified by the access policies are able to decrypt it. 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 keys, and no one else can 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.

A trivial solution to control the access to stored data in a cloud is that the users can employ a symmetric key encryption scheme to encrypt the private data and distribute the decryption key to the users, with whom she or he wishes to share the data, via a secure channel. However, in the real world, there could be a large number of data owners[1] who may want to store their private data in the cloud as well as a large number of users who may want to access the stored data. If a user needs to obtain a key for each owner who wants her or him to access the private data, the number of keys that each user needs to keep could become very large. Furthermore, when the owner wants to revoke the permission from existing users, a new key needs to be distributed to all the other users who have not been revoked to decrypt any future message. This would lead to significant inefficiencies in the implementation of a large-scale system and cause poor performance. Therefore, more sophisticated methods of handling access control

[1] Data owners and users could be ordinary individuals or enterprises or government organisations.

structures are required to simplify security management.

Many access control models have been proposed over the years in the literature. One well-known access control model is the role-based access control (RBAC) model. The central notion of RBAC is that permissions are associated with roles, and users are assigned to appropriate roles. For instance, users are assigned membership to roles based on their responsibilities and qualifications in the organisation. RBAC provides a valuable level of abstraction to promote security management at business function level of an enterprise rather than at the user identity level. It is able to provide a more flexible approach to the management of the users and permissions in contrast to the traditional mandatory access control (MAC) and discretionary access control (DAC), especially when it comes to large-scale systems.

Since users are not assigned permissions directly, similar or related permissions can be grouped together and linked to a role. Hence the issue of granting permissions to users is transformed into the issue of assigning appropriate role(s) to users. This approach not only helps to reduce the effort in mapping the users and permissions, but also separates the operations of the individual users from those of the permissions, which makes the security management in access control systems easier and simpler. Another important feature of RBAC models is role inheritance; that is, roles can inherit permissions from other roles in RBAC models. For example, if role A inherits all the permissions associated with role B, users who are members of role A will be able to access all the permissions that role B has. Role A is referred to as an ancestor role of role B, and role B a descendent role of role A. This hierarchical structure simplifies the task of organising and managing access.

In this thesis, our aim is to propose a comprehensive solution to address security issues in cloud data storage systems using RBAC models to protect data privacy. We have developed secure access controls and trust models for encrypted data storage in a cloud, and have designed architecture for a trust enhanced secure cloud storage system by integrating these proposed models. Based on the designed architecture, we have built and implemented a practical system that can provide secure data storage services in a cloud.

1.2 Problems and Challenges

The security architecture of a distributed system usually consists of three processes: authentication, authorisation and accounting. To consider a secure cloud storage system, it is necessary to address all these processes. Let us now look at the issues in each of these processes, in the context of using RBAC in a cloud storage system.

Before enforcing access control, in general it is necessary to determine the identity of a user who requests access to a resource. The process of authentication involves verification of the identity of a user that it is who it claims to be. Typical authentication methods include passwords and tokens such as smart cards, and biometrics such as iris scans and fingerprints. When a user registers in a system, the administrator of the system will authenticate the user's identity before it allows the user to access the system. We assume that standard authentication mechanisms are available which can be performed by the system to identify the user. Similarly, before a user is included into a role, the manager of the role will need to authenticate the user in order to ensure that the user qualifies for the role. We do not consider the authentication mechanisms between role managers and users; we assume that such mechanisms exist and the role managers will grant role membership only to appropriately qualified users in the system. Since the authentication in the RBAC models can be the same as in other systems, we assume that the systems which we consider will use standard authentication mechanisms for the entities to authenticate each other. In this thesis, we focus on the *authorisation* and *accounting* services in a cloud.

Authorisation or access control refers to a set of security policies which define the users' permissions to access resources in the system. We will use access control and authorisation interchangeably in this thesis and will not enter into a detailed discussion of the differences between these two terms. Accounting typically comes once the resources have been accessed. Access statistics can be used for different purposes such as authorisation control, trend analysis, and capacity planning, which in turn can then be used to further improve security services.

In a cloud data storage system, data owners may wish to specify the policies as to who can access their data and cloud providers are required to correctly enforce these policies. In order to enable the enforcement of the specified access control policies before putting the data into the cloud, the data owners can encrypt the data in such a way that only the users that satisfy the access policies specified by the owners are able to decrypt and access the data. No one else (unqualified users) should be able to reveal the data content. This restriction also applies to the cloud provider itself. That is, the cloud provider itself is subjected to these access controls, thereby ensuring that it does not have access to stored data in its own cloud without the explicit granting of the access privilege by the data owner.

Therefore, to enforce role-based access policies in cloud storage systems, a cryptographic solution is needed to prevent unauthorised access to stored data in the cloud. This cryptographic solution should meet the following high level requirements.

Secure The basic requirement of a cloud storage system is that it should protect the security and privacy of the data stored in the cloud. The system should allow data to be encrypted in such a way that only users, who are allowed by the access policies, can decrypt the data. No one else, including the cloud providers themselves, should be able to reveal the data content.

Flexible Using cryptographic techniques to restrict access control in cloud storage systems transforms the problem of managing access to stored data in the cloud into the problem of managing keys for decrypting the data. The proposed solution should be flexible in terms of both user management and key management when using RBAC models in cloud storage systems.

- The administrator of the system should have the ability to delegate user management operations to the managers of individual roles. That is, the role managers can add/revoke users to/from the role without the need to get the administrator involved.

- In addition, the user management operations of a role manager should not affect users and any other roles in the system. Otherwise, user management operations could become inefficient, especially when there are a large number of users and roles in the system.

- When a user belongs to multiple roles at the same time, the user should be able to use the same key regardless of the number of the roles that she or he belongs to. This brings in the benefit that a user does not need to update the key when her or his role membership changes.
- When the user membership of a role changes, the manager of the role should not need to inform any user of the role about the changes. That is, the role manager should be able to change the user membership without affecting other users in the role.

Efficient A practical cloud storage system should provide good user experience. The better the user experience, the greater the number of users who will be attracted to use the service. We consider the practical ease of use and the computational efficiency of the cloud storage system from two different aspects.

- The setting up of the system should not lead to a significant increase in cost, which is over and above the cost associated with defining RBAC policies. This would be extremely useful when the system has a complex role-based access structure, and when the access policies are often dynamically changed.
- The algorithms involved in carrying out the normal operations of the system should be computationally efficient; this is especially necessary in large-scale systems with a large number of users and roles.

A cryptographic RBAC scheme is a cryptographic approach which integrates an encryption scheme with a RBAC model to enforce the access policies in an untrusted environment. The cryptographic RBAC scheme allows the data to be encrypted to a specific role in the system, and only users who are members of this role, or members of roles that inherit from this role, would be able to access the data by decrypting it. This approach allows data to be encrypted before storing it in an untrusted cloud environment and the stored ciphertext can only be decrypted by those who are allowed by the access policies. Several cryptographic RBAC schemes have been developed to enforce role-based access policies on outsourced data [126, 150]. However, each of these schemes has some problems and none of them satisfies all the requirements mentioned above.

Defining the access policies for large-scale systems is often a time-consuming and error-prone task. In small RBAC systems, a central authority is able to manage users and their permissions. However in large-scale RBAC systems, which have a large number of roles, users and permissions, it becomes impractical to centralise the task of security management with a small group of security administrators. Therefore, decentralising the administration tasks of RBAC systems is a critical design issue when developing large-scale role-based systems. Though several administrative RBAC (ARBAC) models have been developed to achieve decentralised administration of privileges, they are not compatible in managing cryptographic RBAC based systems. Secure enforcement of the administrative polices in ARBAC models to manage cryptographic RBAC systems poses significant challenges. In this thesis, we will be addressing not only the design of cryptographic RBAC schemes but also the decentralised administration of privileges in large-scale cryptographic RBAC systems.

Enforcing RBAC policies using cryptographic approaches ensures that only users with specific roles that are allowed by the data owners can decrypt the data. It is worth noting that the security of a RBAC system using this approach is under the assumption that all the entities behave in a trusted manner so they do not breach the RBAC policies. However, for example, in a cloud storage system that uses RBAC to control the access to the data, an authorised user of the system may leak the data in the cloud to unauthorised users; or an authorised user may be excluded by a malicious administrator of the system from accessing the permissions of the role that have been legitimately assigned to the user. Such issues relate to aspects of trust in these systems.

Trust plays an important role in the accounting process. Based on the analysis of data access history, a trust model can assist in evaluating the trustworthiness of the entities in the system. By utilising the trust evaluation model, the administrator of a system can exclude untrusted entities from the system. Furthermore, one entity in the system can determine whether it is safe to have interactions with another entity. There have been some works [36, 53, 125] which have addressed only the trustworthiness of users in RBAC systems. These trust models assist the RBAC systems in determining the access privileges of users based on their behaviour history. However, there is no existing trust model which considers the trust of the RBAC systems themselves. Trust models are also needed to assist data owners and users to determine the trust of the RBAC system when interacting with it. In addition, considering role hierarchies in the trust models and their evaluation poses significant challenges. In this thesis, we will be developing a comprehensive trust model for cryptographic RBAC systems addressing the various aspects of trust such as users' trust on the RBAC system, RBAC system's trust on the users as well as trust within the RBAC hierarchy.

1.3 Contributions

In this thesis, we propose a comprehensive solution for building a secure RBAC-based cloud storage system. The proposed solution addresses the security issues in the *authorisation* and *accounting* processes of a cloud storage system.

1. We propose three generic constructions of cryptographic RBAC schemes by using ID-based broadcast encryption (IBBE) techniques. We refer to these new constructions as Role-based Encryption (RBE) as it involves the use of cryptography with RBAC to simplify security management while ensuring the enforcement of access policies and the schemes are built directly on RBAC policies. We formalise the definition of the role-based encryption (RBE) scheme based on the security requirements of a RBAC-based cloud storage system. Our constructions show that IBBE techniques can be used in different ways to build RBE schemes with different features. The constructed RBE schemes allow the owner of data to specify a set of roles to which she or he wishes to grant the permission for accessing the data, and encrypt the data in such a way that only the users in these roles can decrypt and view the plain data. In addition, these constructions are able to deal with role hierarchies, whereby roles can inherit permissions from other roles. We further discuss the differences in these RBE constructions, and describe the

suitable scenarios for using individual constructions based on the advantages of each of them.

2. We propose a specific RBE scheme using a broadcast encryption mechanism described in [48]. This scheme provides a concrete cryptographic solution to enforce the RBAC policies on data that is stored in a public cloud. Apart from the features that the generic constructions have, the proposed scheme has additional advantages, such as, that the size of the ciphertext and the size of the decryption key that the user needs to keep are constant, and a user can be revoked from a role without affecting the owners and other users of the same role. We conduct a detailed security analysis of the proposed RBE scheme, and prove that the scheme is secure against adaptive attacks. Our RBE scheme has several superior characteristics compared to others which have been previously proposed such as constant size ciphertext and decryption key, efficient user revocation and user management, and the ability to handle role hierarchies. We also consider design aspects that can be optimised to achieve efficient implementation. We believe that the proposed scheme is suitable for large-scale systems, especially in the context of achieving user centric secure information storage in a cloud computing environment.

3. We then investigate the design of the secure cloud architecture. We first propose an improved RBE scheme with more efficient user revocation. With the improved RBE scheme proposed, there is an assumption that the cloud infrastructure will faithfully execute the scheme. The improved scheme also requires the cloud to return the up-to-date role hierarchy information. We then introduce a secure cloud data storage architecture based on a hybrid cloud infrastructure to overcome the weaknesses of collusion and the timeliness of role hierarchy information mentioned above. 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 role hierarchy and user membership information, and the public cloud is used to store the actual data that is in the

encrypted form. In this architecture, users who wish to share or access the data only interact with the public cloud; there is no access for public users to 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 information about its structure, but also takes full advantage of the public cloud's capacity to securely store large volume of data.

We develop a secure cloud storage system using the improved RBE scheme and the hybrid cloud architecture. Our implementation strategy allows outsourcing part of the decryption computation to the cloud, in which only public parameters are involved; the same strategy also improves the efficiency of user membership management. The most frequently used system operations such as encryption of data by a data owner, decryption of data by a cloud user are 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 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 the decryptor belongs. We optimise 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.

4. To simplify the administration in a large-scale RBAC system, several administrative RBAC (ARBAC) models have been developed to decentralise the administration privileges. A common feature of these models involves managing a RBAC system using RBAC itself. The administration privileges are decentralised to a set of administrative roles in these models, and administrative policies are specified to limit the privileges of administrative roles.

   In order to manage and enforce administrative policies for RBE schemes in large-scale cloud systems, we first introduce a variant RBE scheme in which the role management operations can be decentralised, and we then propose a cryptographic administrative model AdC-RBAC. We show how the proposed model can be used in an untrusted cloud while guaranteeing its security using cryptographic access control enforcement techniques for the proposed variant RBE scheme. The AdC-RBAC model uses cryptographic techniques to ensure that the administrative tasks such as user, permission and role management are performed only by authorised administrative roles. Any other party, including the cloud providers themselves, cannot change the RBAC systems and policies. We describe three components in this model: UAM for user membership management, PAM for permission management, and RAM for role management.

5. RBAC has been widely used in many systems to provide users with flexible access control management. Depending on which party specifies the access policies, the cloud storage systems can be categorised into two different types: admin-centric and user-centric. In an admin-centric system, access permissions are managed by administrators of the system, and all the resources are created and owned by administration team of the storage system. A user-centric system allows data to be managed by individual users who own the data. Users store their own data in the storage system, and specify the access policies as to who can access their data.

   We discuss application examples from both these two different types of systems and show how our proposed system can be used to protect the data privacy in cloud storage systems. In the admin-centric case, we describe the example of an end user banking organisation which uses a RBAC model to control the documents distribution through a cloud platform. For the user-centric case, we discuss the popular scenario where RBAC models are used for storing and sharing of health information. In this case, we consider a patient-centric cloud storage system for personal health records (PHR) data. In both examples, we show how our proposed RBE scheme can enhance the security and privacy of the cloud storage system.

6. Use of the proposed RBE schemes in a cloud storage system can ensure that the stored data is secure under the assumption that all the entities are trusted to behave properly. However, this may not be true in some cases. For example, from a data owner's perspective, a misbehaved role manager may refuse to give the access permission to the owner's data to a user even though the user qualifies for the role, or a malicious qualified user may leak an owner's data to other unauthorised users.

   We propose trust models which can be used to enhance the security of the stored data in cloud storage systems that use the proposed RBE schemes. The trust models provide an approach for the owners and role managers to determine the trustworthiness of individual roles and users respectively. The data owners can use the trust models to decide whether to store their encrypted data in the cloud for a particular role. The roles can use the trust models in their decision to ensure that only users with good behaviour are granted memberships to the roles. The proposed trust models take into account role inheritance and hierarchy in the evaluation of trustworthiness of roles. In addition, we present the design of a trust-based cloud storage system which shows how the proposed trust models can be integrated into a system that uses RBE schemes.

7. Considering the trust of the RBAC systems from the user's perspective, a malicious role manager may refuse to grant the role membership to a user even if the user qualifies for the role, or data assigned to a role from an owner could be infected with virus. We propose trust models for users to address the missing aspect of trust in RBE schemes to improve the decision making for entities (users and role managers) in the cloud system. The proposed trust models can assist (i) the users to evaluate the trust on the roles in a RBAC system and use this trust evaluation to decide whether to join a particular role or not, and (ii) the role managers to evaluate the trust on the data owners in the RBAC system and use this trust in the decision to accept data from an owner. These trust models can not only prevent users from joining roles which have bad historical behaviour

in terms of sharing poor quality resources or misleading users on the content of resources, but also assist the role managers to identify the malicious owners who have caused bad impact on the roles' trustworthiness.

1.4 Structure of the Thesis

The thesis is organised as follows.

Chapter 2 briefly reviews several well-known access control models and the cryptographic approaches which can be used to enforce access policies for these access control models on outsourced data. It extends the discussion on RBAC models and presents several existing cryptographic schemes in literature for access policy enforcement in RBAC models. Then it reviews the concept of trust management, and describes several different types of trust models including the Bayesian trust model which will be used to build trust models later in the thesis.

Chapter 3 defines the formulation and the security properties of role-based encryption (RBE) schemes. It gives three generic RBE constructions by making use of ID-based broadcast encryption schemes, and analyses the security of these RBE constructions. Parts of the chapter appear in [143].

Chapter 4 proposes a specific RBE scheme using an ID-based broadcast encryption mechanism (described in [48]). It describes the security analysis of the proposed scheme and derives proofs showing that the proposed scheme is secure against attacks. It also analyses the efficiency and performance of the proposed scheme and shows that it has superior characteristics compared to other previously published schemes. Parts of the chapter have been published in [142].

Chapter 5 proposes an improved RBE scheme which has an efficient user revocation. Based on the proposed scheme, it presents 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. It describes a practical implementation of the proposed RBE based architecture, and discusses the performance results. Parts of the chapter have been published in [141].

Chapter 6 proposes a cryptographic administrative model, AdC-RBAC, to manage and enforce role-based access policies for RBE schemes, and shows how the AdC-RBAC model decentralises the administrative tasks in a variant of the RBE scheme in Chapter 5, thereby making it practical for security policy management in large-scale cloud systems. The AdC-RBAC model consists of three components, namely the User Administration Model (UAM), the Permission Administration Model (PAM), and the Role Administration Model (RAM). Parts of the chapter appear in [147, 145].

Chapter 7 considers two application scenarios where the proposed RBE schemes can be used to protect the security and privacy of the data stored in cloud systems. It describes an admin-centric banking system which uses RBAC model for the purpose of sharing documents using the cloud, and a user-centric healthcare system that uses RBAC model to manage patient health records stored in the cloud. It shows how the RBE schemes can be used to provide solutions to both these example applications to share data flexibly and protect the sensitive data from being accessed by unauthorised parties.

Chapter 8 proposes the trust models for securing data storage in cloud storage systems that are using RBE schemes. The trust models provide an approach for the owners and roles to determine the trustworthiness of individual roles and users respectively. In addition, it presents a design of a trust-based cloud storage system which shows how the trust models can be integrated into a system that uses RBE schemes. It also describes the relevance of the proposed trust models by considering practical application scenarios and illustrates how the trust evaluations can be used to reduce the risks and enhance the quality of decision making by data owners and roles of cloud storage service. Parts of the chapter have been published in [144].

Chapter 9 proposes the trust models for the users and roles to determine the trustworthiness of individual roles and owners in cloud storage systems that are using RBE schemes. It presents architecture of a trust-based cloud storage system and discusses an application of the proposed trust models to show how the trust models can be used to enhance the security decision making processes. Parts of the chapter have been published in [146].

Chapter 10 summarises the contributions and discusses some possible further work that can be carried out in the future.

2 Background

In a cloud storage system, depending on the security requirements, access policies can be specified in many different ways which can be classified as different access control models. This chapter first reviews several well-known access control models, and then discusses the cryptographic approaches which can be used to secure the data storage in the systems that use these access control models. Following that, several specific cryptographic approaches for the RBAC models are further discussed. Then we review the concept of trust models and describe several commonly used probabilistic models.

This chapter is organised as follows. Section 2.1 briefly describes four well-known access control models. In section 2.2, we discuss the existing cryptographic approaches which can be used to enforce access control policies in there access control models. Section 2.3 extends the discussions on cryptographic RBAC schemes, and reviews several cryptographic schemes which can be used for RBAC models. In section 2.4, we review the concept of trust models and describe the Bayesian trust model. Section 2.5 concludes the chapter.

2.1 Access Control Models

Access control has been widely used by data storage systems in the evaluation of whether a user has access to a particular resource in the system. In a storage system, the stored data needs to be protected from unauthorised access, and the system is expected to control the access to the data according to specific security context and policies that are defined for the storage system. In access control models, the entities that perform the access are referred as subjects, and the resources to be accessed are called objects. Depending on the way that the security policies are specified, access control can be categorised into different models. We first briefly describe several well-known access control models as follows.

2.1.1 Mandatory Access Control (MAC) Model

In a system that uses the mandatory access control (MAC) model, access policies are specified by a central security administrator in the system. There is no concept of individual ownership in the MAC model; all resources are controlled by the system and subject to the MAC policies, and the central administrator(s) decides who can access the resources in the system. Typically subjects (users) in the system are allocated security labels, referred to as security clearances, and objects in the system are allocated security labels, referred to as security classifications. To access a resource in a MAC-based system, the subject must hold the proper security clearance required for that resource with its security classification. The security policy defines rules on the security labels, that is security clearances and classifications. If these rules are satisfied, then the access is allowed.

MAC systems are developed based on the lattice-based access control introduced in [51] where the security is classified into multiple levels and the information flow among the different levels is determined by the mathematical structure of the lattice.

The concept of mandatory access models has been developed and formalised by Bell and Lapadula [11, 99], and the Bell-LaPadula model is the most widely used MAC model. While the Bell-LaPadula model only assures confidentiality of information flows, another model was developed by Biba [13] which deals with integrity only.

When using the MAC model in a commercial organisation, conflicts of interest may arise if a single user simultaneously has access to information from two companies which are direct competitors. The Chinese-wall policy (CWP) was developed by Brewer and Nash [30] to prevent the occurrence of conflicts of interest in a MAC system. Sandhu developed a scheme in which he shows how CWP is mapped to a lattice-based access-control model [116, 117].

The MAC model is considered to be the strictest access control model, and it has been widely used by the military and intelligence agencies to manage their classification access restrictions. However, there are some limitations of the MAC model. Firstly, implementing a MAC system requires significant amount of efforts and expenses due to the reliance on trusted components. After a MAC system is implemented, the overheads to maintain the system can also be very high as the security labels in the system need to be updated for any change in the categorisation and classification of entities in the system.

2.1.2 Discretionary Access Control (DAC) Model

Discretionary access control is a user-centric access control model. It implements a generalised form of access control known as the access-matrix model. This model uses an access matrix for specifying access policies. The concept of the access matrix was introduced in [89, 90], followed by the works in [68, 67]. The access matrix model consists of three types of components, subjects, objects, and access rights. The two-dimensional matrix represents protection states by having a row for every subject, a column for every object, and a flag to indicate the protection state. When a subject requests a resource/object in a DAC system, the system simply looks up the access matrix for both the subject and the object, and grants the access if the flag indicates "allowed".

In contrast to the MAC model, resources in the system governed by the DAC model have owners. These owners have the control over the access permissions to the resources and can determine which users are allowed to access their resources. Since the access permission to a resource is solely specified by its owner, defining security access policies in a DAC model can be easy to implement and hence common in practice. For example, it has been integrated into UNIX operating systems to specify file access permissions where each file can have a set of three permissions, read, write, or execute, to be granted to either a user, group, or others.

A primary benefit of using the DAC model is enabling fine-grained control over system objects. However, the size of the access control matrix can be very large when there are a large number of subjects and objects, as it creates a row and a column for each subject and object respectively. A large amount of space would be wasted especially when the matrix is not dense, for example, when most of the access permissions are set to "disallowed", and the lookup in the matrix would also be expensive in such a case. Thus, DAC access settings are usually stored as either file permission modes such as on UNIX or as access lists instead of a matrix.

Although the DAC model provides huge flexibility for users to specify access control policies, it has several limitations. Since individual users can define the access policies on their data regardless of whether the policies would be inconsistent with the global policies specified by the system, it is very difficult for the system to conduct high level management of the access policies in the system. In addition, the DAC model allows information in one object to be copied to another. Hence, even if the owner of an object does not want to give access to the object to a subject, the subject may still access the object from a copy of it.

2.1.3 Attribute-based Access Control (ABAC) Model

Since the late 1990s, attribute-based access control (ABAC) has emerged with the development of distributed systems. In ABAC models, access permissions to resources are assigned to a set of attributes instead of individual users. Users who satisfy the attributes set can access the resource. In ABAC models, attributes are associated with characteristics of users. The attributes do not necessarily need to relate to each other, and the access policies are defined using a combination of attributes with certain logical relations where a policy can be combined by unlimited number of attributes.

ABAC has been described in various ways [130, 137, 46], and it plays an important role in service-oriented architecture (SOA) and has been used as a standard in web service security specification such as extensible access control markup language (XACML) [106] and security assertion markup language (SAML) [107].

An ABAC system can provide an efficient administration. ABAC allows an access policy to be specified without prior knowledge of a specific subject and there is no limit on the number of subjects that can require access. A major difference between ABAC and other access control models is that the administrator who defines the access policies does not need to specify the capabilities of each individual subject directly. The access control decision is made when a subject makes a request to access a resource in the system. The ABAC model evaluates the attributes set of the subject and grants the access if they satisfy the access policy.

On the other hand, the development and maintenance cost can be higher than simpler access control systems due to the complexity in building an authorisation infrastructure, a large number of subject attributes, as well as additional system support. A careful design requires balancing between the cost of risk and the cost of security. Some systems may have the requirement to periodically review the capabilities associated with subjects and access control entries associated with objects. Since the access each subject has is determined only when it makes a request, the overhead to conduct these audits can be very high as the system may need to get every object owner to run a simulation of the access control request for every known subject in the system.

2.1.4 Role-based Access Control (RBAC) Model

With role-based access control, access decisions are based on the roles that individual users have been assigned to. Users are granted membership to roles based on their competencies and responsibilities in the organisation. Access rights are grouped by role name, and the use of resources is restricted to individuals authorised to the associated role. The use of roles to control access can be an effective means for developing and enforcing enterprise-specific security policies, and for streamlining the security management process. User membership in roles can be revoked easily and new memberships can be established as needed. Role associations can be established when new operations are required, and old operations can be deleted as organisational functions change and evolve. This simplifies the administration and management of permissions; roles can be updated without updating the permissions for every user on an individual basis.

The RBAC model was formally introduced in 1992 [54]. In this model, a role can inherit permissions from other roles. A user who has been granted membership to a role has access to permissions of this role as well as other roles that this role inherits permissions from. The RBAC model was extended and updated in 1996 [120], and the RBAC standard was proposed in 2000 [118]. Four different variations of the RBAC model are defined in [118], namely flat RBAC, hierarchical RBAC, constrained RBAC and symmetric RBAC. The last two types are related to the administration of RBAC systems, and hierarchical RBAC is a more general version of flat RBAC. So in this thesis, when we use the term RBAC, we are referring to a hierarchical RBAC system.

RBAC provides a valuable level of abstraction to promote security management at the business function level of an enterprise rather than at the user identity level. It is able to provide a more flexible approach to management of users and permissions in contrast to the traditional MAC and DAC models, especially when it comes to large-scale systems. Since users are not assigned permissions directly, similar or related permissions can be grouped together and linked to a role. Hence the issue of granting permissions to users is transformed into the issue of assigning appropriate role(s) to users. This approach not only helps to reduce the effort in mapping the users and permissions, but also separates the operations on the individual users from those on the permissions, which makes the security management in access control systems easier and simpler. Another important feature of the RBAC models is the role inheritance; that is, roles can inherit permissions from other roles in RBAC models. This hierarchical structure simplifies the task of organising and managing access even further.

RBAC and ABAC

RBAC and ABAC are similar, and it is easy to transform one model to the other. RBAC can be used as an ABAC model by attaching the role names as attributes and labels to the user members of each role and the objects that each role has permissions to access respectively. An ABAC model can be used as RBAC by attaching attributes to roles, but extra policies need to be specified to define the modes of accessing the protected objects. A more common approach is to use attributes in RBAC models, and a new RBAC standard [77] has been set which allows attributes to be used as constraints for access control decisions, which could bring significant flexibilities to existing RBAC models.

RBAC and ABAC are different, as they have their own advantages and disadvantages. In a dynamic RBAC system, role management could be complex as constraints may be required in a rapidly changing environment. Moreover, the permissions granted to a user through roles need to be evaluated to determine if the permission assignment will be allowed. ABAC provides a more flexible approach and uses objects' label and users' attributes to control access instead of permissions. In addition, ABAC is suitable for distributed environment as the attributes can be managed by different authorities. However, attributes can be difficult to manage if there is a large number of them, and it is usually impractical to audit which users have access to a given permission and what permissions have been granted to a given user while auditing users' permissions is easy in a RBAC model.

2.2 Cryptographic Access Control Schemes

In traditional access control systems, enforcement of access policies 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 keys, and no one else will be able to recover the data content. Therefore, the problem of managing access to data stored in the cloud is transformed into the problem of the management of keys, which in turn is determined by the access policies.

2.2.1 Broadcast Encryption

A trivial solution to protect the privacy of data stored in cloud is to use a cryptographic encryption scheme to encrypt the data before storing it in the cloud. This would allow only the users who have access to the key(s) to decrypt the data and to view the data in its plain form. The problem of achieving secure access to data stored in the cloud is transformed into the problem of access to keys. This approach is suitable for MAC and DAC models, whose access policies can be represented by an access control matrix (ACM).

Table 2.1 shows an example of such an access control model. Let us assume that the matrix is for a cloud storage system using a DAC model. The set $\{f_1, f_2, f_3\}$ represents all the objects in the model, that is, say files stored in the cloud. The set $\{u_1, u_2, u_3\}$ represents all the subjects, that is, the users who want to access these files stored in the cloud. Each file in the model has an owner, and the owners have the flexibility to control who can access the files. Each row in the matrix is a capability list (CL) of the subject, and the column corresponding to each object is called an access-control list (ACL) for that object. A snapshot of the access matrix represents a protection state where 1 means “has access” and 0 indicates “no access”. It is clear that the owner of each file can simply employ a secret key encryption scheme to encrypt the data and distribute the secret key to the users with whom she or he wishes to share the data, and store her or his resource in the encrypted form to the cloud. For example, the

	f_1	f_2	f_3
u_1	1	1	1
u_2	0	1	0
u_3	1	1	0

Table 2.1: Access Control Matrix Example

owner of the file f_1 encrypts and uploads the file and gives the secret key to the users u_1, u_3. Then only u_1 and u_3 can decrypt the file f_1 because they possess the secret key corresponding to the encryption, and no one else can recover the content of the file. We require a secure way of achieving key distribution to these selected users, who have the access to view the data according to the access control policies.

Since the secret key encryption requires different keys to encrypt different objects, the number of keys will become large when there is a massive amount of resources in the system. Therefore the owners may wish to use public key encryption techniques to protect the privacy of their files as they can simply use the public keys of users to encrypt data and do not need to transfer any key to users. However, if an owner uses a public/private key pair to encrypt/decrypt the file, she or he may need to encrypt the same file multiple times if she or he wants more than one users to access the file because the public keys are different for different users. This will make the approach impractical when there are a large number of users in the system. Fortunately, owners can use broadcast encryption schemes to encrypt files in this scenario.

The concept of Broadcast Encryption (BE) was introduced by Fiat and Naor in [56]. In BE schemes, a broadcaster encrypts messages and transmits them to a group of users who are listening in a broadcast channel. Then they use their private keys to decrypt the transmissions. While encrypting the messages, the broadcaster can choose the set of users that is allowed to decrypt the messages. Following this original scheme, many other BE schemes have been proposed [58, 65, 22]. These schemes require public parameters for every user, and every time a user wants to join or leave the system, the public parameters need to be updated.

In the example shown in Table 2.1, the owner of f_1 can use a broadcast encryption scheme to encrypt the file to both u_1 and u_3 using their public keys and uploads the encrypted file to the cloud. Then u_1 and u_3 will be able to decrypt the file using their own private keys. This approach needs a public-key infrastructure (PKI) to manage the public keys of all the users in the system. This is needed for the owners to ensure that they are using the correct public keys to encrypt their files for the right users. However, the system can be made even simpler by using ID-based cryptographic techniques.

In 1984, Shamir [123] suggested the possibility of a public key encryption scheme in which the public key can be an arbitrary string. In 2001, Boneh and Franklin introduced an ID-based encryption (IBE) scheme, in which the sender can use the identity of the receiver as the public key to encrypt the messages. An ID-based broadcast encryption scheme (IBBE) is defined in a similar way. In an IBBE scheme, the system does not need to have any pre-set parameters for every user, and a broadcaster only needs to know the identity of the user if this user is allowed to decrypt the messages. In this case, one user joining or leaving the system will not affect any other user. Moreover, the users do not even need to have the decryption key at the time when the messages were encrypted. They can obtain their keys afterwards. Several IBBE schemes have been proposed subsequently [49, 23, 73].

Generally, an IBBE scheme involves three different parties: a Private Key Generator (PKG), the users with unique identities, and the broadcasters who possess the messages. The PKG generates decryption keys for each authorised users based on her or his identities. A broadcaster can encrypt messages to a selected group of users and transmit the messages via a broadcast channel. The broadcaster uses only the public key and users' identities to encrypt the messages. More formally, an IBBE scheme is composed of four algorithms which are described as follows:

$\mathcal{IBBE}.\mathcal{Setup}(\lambda)$**:** takes as input the security parameter λ and outputs a master key mk and a group public key pk. mk is given to PKG, and pk is made public.

$\mathcal{IBBE}.\mathcal{Extract}(\mathsf{mk}, \mathsf{ID})$**:** an algorithm executed by the PKG, on input of a user identity ID and the master secret mk, returns the user private key $\mathsf{sk}_{\mathsf{ID}}$.

$\mathcal{IBBE}.\mathcal{Encrypt}(\mathsf{pk}, \mathcal{U}, M)$**:** an algorithm executed by the broadcaster, on input of the set $\mathcal{U}$ of identities of users to whom it wishes to encrypt the message M and the public key pk, outputs a pair $\{\mathsf{Hdr}_{\mathcal{U}}, K_{\mathcal{U}}\}$, where $\mathsf{Hdr}_{\mathcal{U}}$ is called the header and $K_{\mathcal{U}}$ is in the key space of a symmetric encryption scheme $\mathcal{E}_{sym}$ (that is, the generated value $K_{\mathcal{U}}$ can be directly used as a key of the encryption scheme $\mathcal{E}_{sym}$).

Assume that a broadcaster wishes to encrypt a message M to a group $\mathcal{U}$ of users with identities $\{\mathsf{ID}_1, \cdots, \mathsf{ID}_n\}$. Let $\mathcal{E}$ be the IBBE scheme and $\mathcal{E}_{sym}^{key}(M)$ be the encryption of M using $\mathcal{E}_{sym}$ under the secret key key. The ciphertext C is denoted as:

$$C = \mathcal{E}_{\mathcal{IBBE}}(M, \mathcal{U}) = \{\mathsf{Hdr}_{\mathcal{U}}, \mathcal{E}_{sym}^{K_{\mathcal{U}}}(M)\}$$

$\mathcal{IBBE}.\mathcal{D}ecrypt(\mathsf{pk}, \mathsf{sk}, C)$**:** an algorithm executed by the user to decrypt the ciphertext on input of the user secret key sk and public key pk.

This algorithm has two steps: the first step takes as input the user secret key sk and the header Hdr, and recovers the value K, and then the second step uses the symmetric key K to decrypt M from $\mathcal{E}_{sym}^{K}(M)$.

Using an IBBE scheme, the owners of the files can encrypt their files using the identities of the users with whom they wish to share the files and upload the encrypted files to cloud. The certificate authorities in the system generate the private keys for users in the system and distribute the private keys to users via a secure channel. The private keys for users can be issued after the files have been uploaded to the cloud.

2.2.2 Attribute-based Encryption

Integration of cryptographic techniques with the ABAC model has led to a technique called attribute-based encryption (ABE). In an ABE system, the access permission to a resource is associated with a set of attributes and a security policy based on these attributes. Only the users who hold the keys for the attributes are able to decrypt the content. This feature allows ABE schemes to be used in protecting the privacy of resources in a cloud storage system which uses the ABAC model to control access privileges.

Goyal *et al.* [63] proposed the first ABE scheme, in which ciphertexts are labelled with sets of attributes and private keys are associated with access structures that control which ciphertexts a user is able to decrypt. Hence this scheme is also referred to as the key-policy ABE or KP-ABE. In the KP-ABE scheme, the owner of the data does not have the control over who is allowed to access the data. The owner must trust that the key-issuer who issues the appropriate keys will grant or deny access to the appropriate users. Bethencourt *et al.* [12] introduced another form of ABE scheme which works in the reverse manner where the user keys are associated with sets of attributes and the ciphertexts are associated with the policies. Hence it is referred to as the ciphertext-policy ABE or CP-ABE scheme. Both algorithms associated a set of expressively descriptive attributes with tree-access structures to enforce access control on the encrypted data, but how the keys and ciphertexts are associated with the access policies works in a reverse manner. As a result, in the KP-ABE scheme, it is the key distributor (usually the service provider), who decides the access policy, while in the CP-ABE scheme, it is the encryptor (usually the data owner) who controls the access over the encrypted data.

The access policies of a basic ABE are specified in a tree structure. Each leaf node corresponds to an attribute defined in the system. Each non-leaf node represents a threshold gate which connects its children attributes or threshold gates. We denote n_x as the number of child nodes that a threshold gate x has and k_x as its threshold value where $0 < k_x \leq n_x$. It is clear that the threshold gate is an OR gate when $k_x = 1$ and it is an AND gate when $k_x = n_x$. Verifying whether a set of attributes S satisfies the access tree $\mathcal{T}$ is a recursive process. We denote $\mathcal{T}(x) = 1$ if and only if the node x is satisfied. When x is a leaf node, $\mathcal{T}(x)$ returns 1 if the attribute associated to the node is in the set S. When x is a non-leaf node, $\mathcal{T}(x)$ returns 1 if at least k_x child nodes are evaluated as 1. Assume the root node of the access structure is r. The access tree is satisfied if and only if $\mathcal{T}(r) = 1$.

Let us look at an example shown in Figure 2.1. Assume that a recruitment agency wants to store the profiles of its job candidates in the cloud so the information can be easily shared with their client companies. Now the marketing department of a company is looking for a salesperson, and they want to access the profiles from the

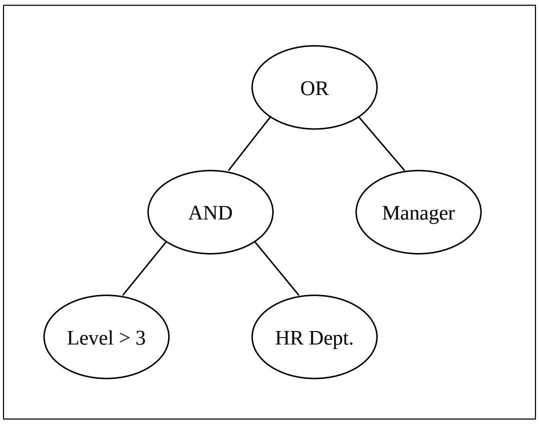


Figure 2.1: Attribute-based Encryption Access Structure Example

recruitment agency. The recruitment agency specifies in its access policies for the company that only the managers of the marketing department or staff from the HR department whose level is greater than three can view the profiles. The recruitment agency can use an ABE scheme to encrypt the suitable profiles and store them in the cloud. Three decryption keys will be generated in this example for the three attributes respectively, and they will be distributed to the employees who qualify the attributes. The managers of the marketing department will be given the key for the attribute "Manager", all the staff in the HR department will be given the key for the attribute "HR Dept.", and only the staff whose levels are greater than three will be given the key for the attribute "Level >3". Then a staff of the company will be able to decrypt and view the candidates' profiles with the given keys if and only if the attributes associated with her or his keys satisfy the access tree.

When using a KP-ABE scheme to encrypt resources, attributes are assigned to the ciphertext in the encryption, and the policies for the access structure are associated with the decryption keys when the keys are generated. It is the authority who generates the keys that decides the access policies. In the above example, KP-ABE is suitable in the scenario where the profiles of candidates will be shared with another client company which has a different organisational structure. Then ciphertext of the profiles are associated with the same set of attributes, but the decryption keys will be generated separately under the different access policies of the other company.

In a CP-ABE scheme, the access policies are associated with the ciphertext and are specified in the encryption. Keys can be generated prior to the data encryption, and remain unchanged when data is encrypted under different access policies. In the above example, CP-ABE is suitable in the case where the profiles need to controlled under different policies in the same client company. For example, the profiles of the sales candidates are encrypted so that only the managers of the marketing department can view them, and the profiles of the technician candidates are allowed to be viewed only by the managers of the technical department. Then the employees of the client company only need to hold a single key for an attribute even it is used in several different access structures. The recruitment agency only needs to specify the different access policies while encrypting different profiles.

A typical ABE scheme consists of following four algorithms: *Setup*, *Encrypt*, *KeyGen*, and *Decrypt*. However, the input parameters of the algorithms *Encrypt* and *KeyGen* are different in KP-ABE and CP-ABE because the access structures are associated with keys and ciphertexts respectively in these two types of ABE schemes. Let us now describe the algorithms of an ABE scheme as follows:

$\mathcal{ABE}.\mathcal{S}etup(\lambda)$: takes as input the security parameter λ and outputs a master secret key mk and a public key pk. mk is kept secretly, and pk is made public.

$\mathcal{KP}\text{-}\mathcal{ABE}.\mathcal{E}ncrypt(\mathsf{pk}, M, S)$: an algorithm on input of the system public key pk, a message M and a set of attributes S, outputs the ciphertext C. In this algorithm, only the attributes are used in computing the ciphertext, and the access structure of the attributes is not specified. Not all the attributes need to appear in the access structure, but S needs to be the super set of all the attributes which will be used in the access structure.

$\mathcal{KP}\text{-}\mathcal{ABE}.\mathcal{K}eyGen(\mathsf{pk}, \mathsf{mk}, \mathcal{T})$: an algorithm on input of the system public key pk, master secret key mk and the access structure T, outputs a set of decryption keys dk.

$\mathcal{CP}\text{-}\mathcal{ABE}.\mathcal{E}ncrypt(\mathsf{pk}, M, \mathcal{T})$: an algorithm on input of the system public key pk, a message M and the access structure T, outputs the ciphertext C.

$\mathcal{CP\text{-}ABE.KeyGen}(\mathsf{pk}, \mathsf{mk}, S)$**:** an algorithm on input of the system public key pk, a message M and a set of attributes S, outputs a set of decryption keys dk. Similar to that in $\mathcal{KP\text{-}ABE.Encrypt}$, the attribute set S is a super set of the attributes in the access structure.

$\mathcal{ABE.Decrypt}(\mathsf{pk}, \mathsf{dk}, C)$**:** an algorithm on input of the system public key pk, the decryption keys dk and the ciphertext C, outputs the message M in the plaintext form.

In the above described ABE schemes, the access policy can only contain the logical formula "and" and "or", and threshold gates. Ostrovsky *et al.* [110] introduced a KP-ABE scheme which allows "negative" constraints to be represented in access policies. Additionally, Many CP-ABE schemes were proposed [39, 76, 52, 92, 131] which are either chosen ciphertext attack (CCA) secure or built on different security assumptions. Even though the KP-ABE and CP-ABE work in reverse manner, Goyal *et al.* [64] provided a generic approach to transform a KP-ABE scheme into a CP-ABE one. Malek and Miri combined the two ABE schemes into one system, and proposed a balanced access control that allows both the service provider setting up system wide access policies and the data owner setting up access structure to control the access to their own data [98]. Further research on ABE is also discussed in [136, 139]. In a dynamic system, access policies may differ from time to time, and user qualifications may also change. Therefore, the ability to revoke attributes from a user is desired in ABE systems. Several revocable ABE schemes [138, 113] were proposed where an ABE system is able to revoke users from accessing encrypted data to which they used to have access in the system.

In an ABE system, a user needs to authenticate to the key generation authority to prove her or his identity to obtain the keys to decrypt the data. When using ABE in a system where there is a large number of attributes, assessing the qualification of users and generating decryption keys by a central authority is usually impractical. *Multi-Authority Attribute-Based Encryption* (MA-ABE) was first proposed to address this issue in 2007 [37]. In a MA-ABE scheme, attributes are divided into different sets, and each set can be managed by an independent attribute authority. Corresponding attribute keys for decryption are issued by multiple attribute authorities, and encryptors can specify an access policy that requires a user to obtain decryption keys for appropriate attributes from different authorities in order to decrypt a message. Subsequently, several other MA-ABE constructions were proposed [95, 38, 91].

2.2.3 Cryptographic RBAC Schemes

A cryptographic RBAC scheme is a cryptographic approach which integrates an encryption scheme with RBAC models to enforce the RBAC policies in an untrusted environment. The cryptographic RBAC scheme allows data to be encrypted to a specific role in the system, and only users who are members of this role or members of roles that inherit from this role would be able to access the data by decrypting it. This approach allows data to be encrypted before storing it in an untrusted cloud environment and the stored ciphertext can only be decrypted by those who are allowed by the access policies. There have been many different types of approaches proposed to consider the security issues for applying RBAC models to data outsourcing scenarios.

Akl and Taylor [1] introduced a hierarchical cryptographic access control scheme in 1983. Because of the similarity in structures between hierarchical access control and RBAC, a hierarchical cryptographic access control scheme can be easily transformed into a cryptographic RBAC scheme. MacKinnon *et al.* [97] proposed an improved solutions to address a few weaknesses in [1]. Instead of using the top-down design approach as in the above two schemes, Harn and Lin [66] presented a scheme using a bottom-up key generating procedure. Since then, a number of approaches [93, 88, 94, 140, 40, 74, 133, 45] have been proposed using cryptographic techniques to enforce access control policies, which are known as key assignment schemes.

The problem of access control for securely outsourcing data using cryptographic techniques was first considered by Miklau and Suciu [102]. An improved scheme was proposed by Vimercati *et al.* [127] to address access policy updates. Several cryptographic access control approaches [5, 69, 151, 17] have been investigated to address the problem of secure data access and cost effective key management in a distributed environment. Subsequently, a two layer encryption model was proposed [126] to prevent a service provider from accessing the content of data, but the service provider is able to run queries or perform other operations on the data for users who can decrypt the data using their keys. The approaches that use similar mechanisms to enforce the access control policies are further discussed in [114, 128]. However, these solutions have several limitations. For instance, if there is a large number of data owners and users involved, the overheads involved in setting up the key infrastructure can be very high. 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 in large-scale systems.

Hierarchical ID-based Encryption (HIBE) is an alternative approach for the management of keys. In the HIBE schemes [59, 20], a user with an identity in the hierarchy tree can decrypt messages encrypted to its descendant identities, but cannot decrypt messages encrypted to other identities. HIBE schemes can be easily used to enforce access policies in RBAC models by associating the leaf nodes in the hierarchy tree with users and non- leaf nodes with roles. However, there are several issues with the HIBE schemes. Firstly, in a HIBE scheme, the length of the identity becomes longer with the growth in the depth of hierarchy. Secondly, 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. Zhu *et al.* [148] introduced a role-based encryption (RBE) scheme which considers a hierarchical RBAC model and allows data to be encrypted in the way that only users with appropriate roles can decrypt. However this scheme lacks the ability of user revocation and the size of the ciphertext increases linearly with the number of all the ancestor roles. Another RBE scheme was then proposed in [150]. In this scheme, once again, the size of the ciphertext increases linearly with the number of all the ancestor roles, and the encryption involves using the identity list of the revoked users.

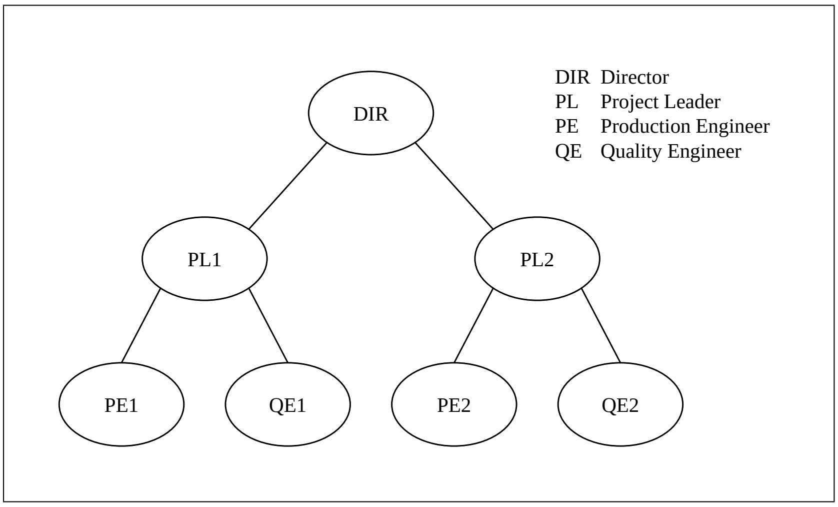


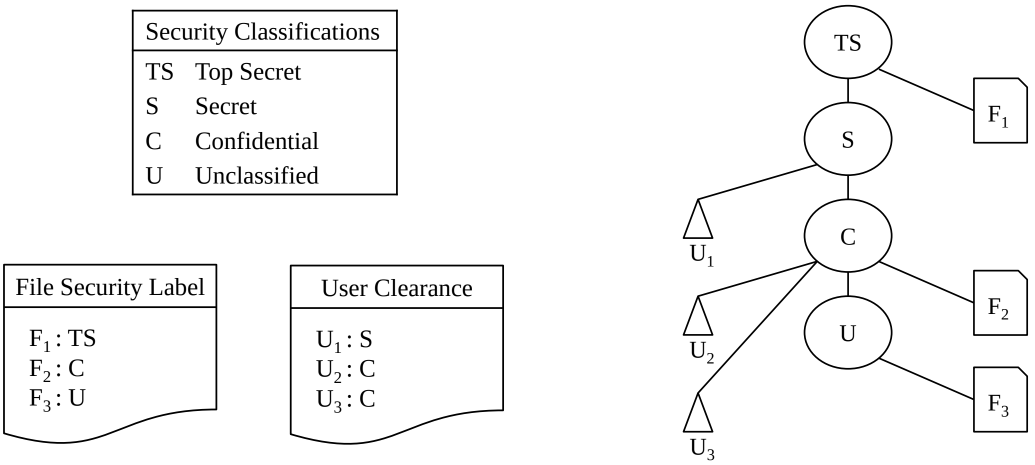
FIGURE 2.2: Hierarchical RBAC Example I

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.

Now we show how a cryptographic RBAC scheme can protect the data privacy in RBAC models by looking at a RBAC example. Assume that Figure 2.2 represents a hierarchical role structure of a department in a company. The department has two projects *PL1* and *PL2*, and each project has two sub-roles *PE* and *QE*. The role *PL* inherits permissions from both roles *PE* and *QE* within the project, and there is a role *DIR* that inherits from both role *PL1* and *PL2*. Now the department wants to store confidential documents in the cloud and uses an cryptographic RBAC scheme to encrypt the documents.

Initially, the administrator sets up the system structure, and generates the keys for existing users, that is, the staff in the department. Assume one user in the role *PL1* has created a document for project 1 and wants to upload it to the cloud for other leaders of the project 1 to review. The user can simply encrypt the document to the role *PL1*, and upload it to the cloud. Other users who are in the role *PL1* can use their own decryption keys to decrypt the document and review. Since the role *DIR* inherits from the role *PL1*, the users in the role *DIR* who are the directors of the department can also decrypt the document using their own decryption keys. Meanwhile, neither the users in the role *PE1*and *QE1*, nor the users in the role *PL2*, *PE2* and *QE2* can decrypt and view the document.

Mandatory Access Control (MAC) Model



(a) MAC example

(b) Cryptographic MAC Model

Figure 2.3: Cryptographic MAC Model Example

In a MAC model, each resource (object) is associated with a security clearance, and each user (subject) is assigned a clearance. The users are only allowed to access the resources whose security classification level is lower than or equal to their clearance level. We consider an example of a MAC model shown in Figure 2.3(a). The system security classifications are defined as a set {*Top Secret (TS), Secret (S), Confidential (C), Unclassified (U)*}. Three files (F_1, F_2, F_3) in the system are associated with the security classification (TS, C, U) respectively, and three users (U_1, U_2, U_3) of the system are assigned the (S, C, C). Because of the information flow control in MAC model, U_1 can access files with the security level (S, C, U), and the users U_2 and U_3 can only access files with the security level (C, U).

Now we transform the access policies into a hierarchical structure, and then use a cryptographic RBAC scheme to enforce the MAC policies. The hierarchy is shown in Figure 2.3(b), where we map the four security classifications to four different roles. We organise the roles following the direction of the information flow propagation, and let the role TS inherits from S, S inherits from C, and C inherits from the role U.

Each file is then encrypted to the role that is associated to its security label using a cryptographic RBAC scheme, and users are granted the membership of the roles corresponding to their clearance levels. Then we can see that U_1 can decrypt the files encrypted to the role (S, C, U), and U_2 and U_3 can only decrypt the files encrypted to the role (C, U). Hence the access policy enforced by the cryptographic RBAC scheme is identical to the MAC policy specified in Figure 2.3 (a). Therefore, one can see that we can use cryptographic RBAC schemes to enforce MAC policies. In general, the cryptographic RBAC schemes are able to be used to enforce the policies of a more complex lattice-based MAC model in a similar way to that we have described above.

2.3 Cryptographic RBAC Approaches

From the above description, we see that several types of cryptographic approaches can be used in cryptographic RBAC schemes. In this section, we present several example cryptographic schemes for each type.

2.3.1 Hierarchical Key Management Schemes

We first look at the hierarchical key management schemes that can be used for role-based access policies enforcement. The security of the hierarchical key management scheme relies on the correct execution of the key assignment process, and the key for the user is generated based on the access control policies of the whole system.

Atallah *et al.*'s Key Management Scheme [5]

This scheme uses a secure symmetric key encryption scheme to encrypt the data, and the key management scheme manages the secret keys of the symmetric key encryption. Each secret key is associated with a node in a hierarchy. Assume that $H : \{0,1\}^* \rightarrow \{0,1\}^n$ is cryptographic hash function, where $\{0,1\}^n$ is the key space of the symmetric key encryption scheme, the key management scheme works as follows:

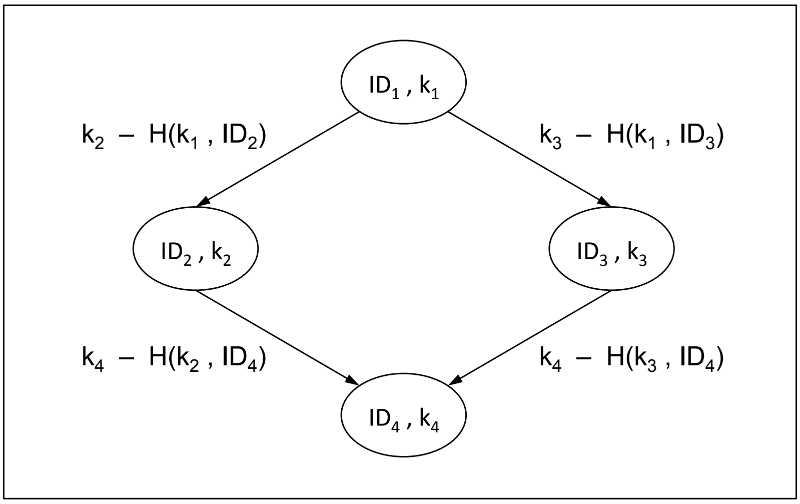


Figure 2.4: Key Allocation for HKM Access Graph Example

Key generation. Assume that each node is an entity in the system. The private key generation process and the nature of public information stored at each node of the hierarchy are as follows:

- **Private key.** Each entity v_i is assigned a random private key $k_i \in \{0,1\}^n$. v_i is given all keys of the other entities whose access levels are lower than v_i.
- **Public information.** For each entity v_i there is a unique identity $\mathsf{ID}_i \in \{0,1\}^*$ that is assigned to the entity. Also for each edge (v_i, v_j), the value $y_{i,j} = k_j - H(k_i, \mathsf{ID}_j) \bmod 2^n$ is stored publicly for this edge.
- **Key derivation.** All that needs to be shown is how to generate a child's key from the parent's private information and the public information. Suppose v_i is a parent of v_j with respective keys k_i and k_j. Since ID_j and $y_{i,j} = k_j - H(k_i, \mathsf{ID}_j) \bmod 2^n$ are public information, node v_i can generate k_j with this information.

Example: Figure 2.4 shows the key allocation for a graph which is more complicated than a tree structure. First, it is possible for the node with k_1 to generate key k_2, because that node can compute $H(k_1, ID_2)$ and use it, along with the public edge information, to obtain k_2. The node with k_3, on the other hand, cannot generate k_2, since this would require inversion of the H function.

This hierarchical key management scheme is the basis of several other schemes [127, 151, 17]. Next we describe the scheme proposed by Vimercati *et al.* [127] which is based on the algorithm described in Atallah *et al.*'s scheme [5].

Vimercati *et al.*'s Key Management Scheme [127]

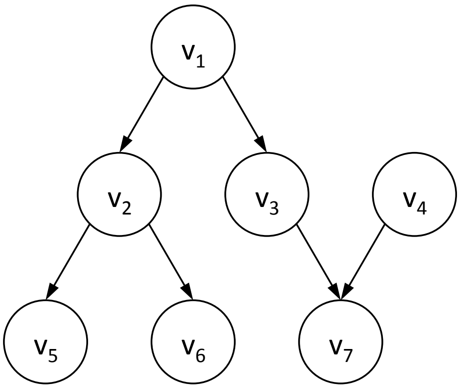
In [127], the authors use the combination of the above hierarchical key management scheme and a two-layer encryption to protect the privacy of the outsourced data. This scheme uses two-layer encryption to prevent a service provider from accessing the content of data while the service provider is able to run queries or perform other operations on the data for users who can decrypt the data using their keys. Now we describe the scheme as follows:

Let $\mathcal{K}$ be the set of symmetric encryption keys in the system, and $\mathcal{T}$ the set of tokens in the public catalog.

- The direct key derivation function $\tau : \mathcal{K} \rightarrow 2^{\mathcal{K}}$ is defined as $\tau(k_i) = \{k_j \in \mathcal{K} \mid \exists t_{i,j} \in \mathcal{T}\}$.
- The key derivation function $\tau^* : \mathcal{K} \rightarrow 2^{\mathcal{K}}$ is a function such that $\tau^*(k_i)$ is the set of keys derivable from k_i by chains of tokens, including the key itself (chain of length 0).

Keys and tokens can be graphically represented through a graph, which contains a vertex v_i associated with each key in the system, denoted k_i, and an edge connecting two vertices (v_i, v_j) if token $t_{i,j}$ belongs to the public token catalogue $\mathcal{T}$. Chains of tokens then correspond to paths in the graph. For example, Figure 2.5 (a) represents an example of a graph corresponding to a set of seven keys, along with its public token catalogue composed of six items (see Figure 2.5 (b)). As an example, $\tau(k_1) = \{k_2, k_3\}$ and $\tau^*(k_1) = \{k_2, k_3, k_5, k_6, k_7\}$.

Let $\mathcal{U}$ be the set of users of the system and $\mathcal{R}$ the set of outsourced resources. Authorisations can be modelled via a traditional access matrix $\mathcal{A}$, with a row for each user $u \in \mathcal{U}$, a column for each resource $r \in \mathcal{R}$. Each entry $\mathcal{A}[u, r]$ is set to 1 if u can



(a)

Source	Destination	Value
v_1	v_2	$k_2 \oplus H(k_1, \mathsf{ID}_2)$
v_1	v_3	$k_3 \oplus H(k_1, \mathsf{ID}_3)$
v_2	v_5	$k_5 \oplus H(k_2, \mathsf{ID}_5)$
v_2	v_6	$k_6 \oplus H(k_2, \mathsf{ID}_6)$
v_3	v_7	$k_7 \oplus H(k_3, \mathsf{ID}_7)$
v_4	v_7	$k_7 \oplus H(k_4, \mathsf{ID}_7)$

(b)

Figure 2.5: Key Derivation for HKM Example

	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
A	1	1	1	0	0	1	1	1	1
B	0	0	0	0	0	1	0	0	0
C	0	0	1	0	0	0	1	1	1
D	0	0	0	1	1	0	1	1	1
E	0	0	0	0	0	1	0	0	0

(a)

u	$\phi(u)$
A	k_1
B	k_2
C	k_3
D	k_4
E	k_5

(b)

r	$\phi(r)$
r_1, r_2	k_1
r_3	k_3
r_4, r_5	k_4
r_6	k_5
r_7, r_8, r_9	k_7

(c)

Figure 2.6: Access Matrix and Key Assignment for HKM Example

access r; 0 otherwise. Figure 2.6 (a) illustrates an example of an access matrix with five users (A, B, C, D, E) and nine resources $(r_1, r_2, \cdots, r_9)$. Given an access matrix $\mathcal{A}$ over sets $\mathcal{U}$ and $\mathcal{R}$, $acl(r)$ denotes the access control list of r (i.e., the set of users that can access r). For instance, with reference to the matrix in Figure 2.6 (a), $acl(r_1) = A$.

Assume that each user is associated with a single key, communicated to her or him by the owner on a secure channel at the time the user joins the system. Also, each resource can be encrypted by using a single key. A key assignment is a function $\phi : \mathcal{U} \cup \mathcal{R} \rightarrow K$, which associates each user $u \in \mathcal{U}$ to whom the (single) key released and each resource $r \in \mathcal{R}$ to the (single) key with which the resource is encrypted.

From the previously described scheme, it is easy to see that each user u can retrieve (via her or his own key $\phi(u)$ and the set of public tokens $\mathcal{T}$) all the keys derivable from $\phi(u)$, that is, all the keys in $\tau^*(\phi(u))$.

Figure 2.6 represents an example of a key assignment, with reference to the key graph in Figure 2.5 (a). Composing the graph in Figure 2.5 (a) with the function defined in Figure 2.6 (b), the set of keys each user knows is obtained. As an example, $\phi^*(B) = \tau^*(k_2) = \{k_2, k_5, k_6\}$.

As an example, consider the access policy in Figure 2.6 (a). Since there is a set of five different *acls* (i.e., $\{A\}$, $\{A, C\}$, $\{D\}$, $\{A, B, E\}$, $\{A, C, D\}$), which includes a singleton *acl*, and five users, seven different keys need to be defined. In particular, keys $k_1, \cdots, k_5$ are associated, in the order, with users $A, \cdots, E$, k_6 is associated with A, B, and k_7 with A, C, D. Then a token is defined between each pair of keys (k_i, k_j), $i, j = 1, \cdots, 7$ and $i \neq j$, such that the set of users corresponding to k_i is included in the set of users corresponding to k_j . Figure 2.5 (a) illustrates a graphical representation of these keys and tokens, where vertex v_i corresponds to key k_i. Resources $r_1, \cdots, r_8$ are then encrypted as shown in Figure 2.6 (c).

2.3.2 Hierarchical Identity-based Encryption

Hierarchical ID-based Encryption (HIBE) is a generalisation of ID-based Encryption (IBE) that mirrors an organisational hierarchy. In HIBE schemes, identities are organised in a hierarchical structure. A user with an identity in the hierarchy tree can decrypt the messages encrypted to its descendant identities, but cannot decrypt messages for other identities. The concept of Hierarchical ID-based encryption was first introduced by Horwitz and Lynn [72], and Gentry and Silverberg [59] gave the first HIBE construction. In this construction, the length of ciphertexts and private keys grow linearly with the depth of the hierarchy tree. Boneh *et al.* [20] proposed another HIBE scheme where the ciphertext size as well as the decryption cost are independent of the depth of the hierarchical tree.

As the roles can be organised in a hierarchical manner in a RBAC system, it is clear that these roles can be directly mapped to identities in a HIBE scheme. To integrate HIBE into an existing RBAC system, an identity hierarchy of the HIBE scheme can be constructed to be the same as the role hierarchical structure of the RBAC system, with each node in the identity hierarchical tree mapped to the role in RBAC corresponding to the same path. When data owners encrypt their data to identities of roles to which they want to give access, only the ancestor identities (ancestor roles) and the target identities themselves can decrypt the data, thereby enforcing the RBAC policies.

Gentry-Silverberg's HIBE [59]

The first HIBE construction was given in [59]. The entities are organised in a tree structure, and each entity is mapped to an identity of a user. Let $Level_i$ be the set of entities at level i, where $Level_0 = \{\text{Root PKG}\}$. The scheme consists of the following algorithms,

Root Setup: The root PKG chooses two groups $\mathbb{G}_1, \mathbb{G}_T$ of some prime order q, a pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$, a random generator $P_0 \in \mathbb{G}_1$, a random $s_0 \in \mathbb{Z}_q^*$ and set $Q_0 = s_0 P_0$. Then the root PKG picks two cryptographic hash functions $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and $H_2 : \mathbb{G}_T \to \{0,1\}^n$ for some n.

The message space is $\mathcal{M} = \{0,1\}^n$, the root PKG's secret is s_0, and the system parameters are

$$\mathsf{pub} = (\mathbb{G}_1, \mathbb{G}_T, \hat{e}, P_0, Q_0, H_1, H_2)$$

Lower-Level Setup: Entity $E_t \in Level_t$ picks a random secret $s_t \in \mathbb{Z}_q^*$.

Extraction: Let E_t be an entity in $Level_t$ with ID-tuple $(\mathsf{ID}_1, \cdots, \mathsf{ID}_t)$, where $(\mathsf{ID}_1, \cdots, \mathsf{ID}_i)$ for $1 \le i \le t$ is the ID-tuple of E_t's ancestor at $Level_i$. Set S_0 to be the identity element of $\mathbb{G}_1$. Then E_t's ancestor does the following:

1. computes $P_t = H_1(\mathsf{ID}_1, \cdots, \mathsf{ID}_t) \in \mathbb{G}_1$
2. sets E_t's secret point S_t to be $S_{t-1} + s_{t-1}P_t = \sum_{i=1}^{t} s_{i-1}P_i$
3. also gives E_t the values of $Q_i = s_i P_0$ for $1 \le i \le t-1$

Encryption: To encrypt $M \in \mathcal{M}$ with the ID-tuple $(\mathsf{ID}_1, \cdots, \mathsf{ID}_t)$, do the following:

1. computes $P_i = H_1(\mathsf{ID}_1, \cdots, \mathsf{ID}_i) \in \mathbb{G}_1$ for $1 \le i \le t-1$
2. choose a random $r \in \mathbb{Z}_q^*$

3. set the ciphertext to be

$$C = [rP_0, rP_2, \cdots, rP_t, M \oplus H_2(g^r)] \text{ where } g = \hat{e}(Q_0, P_1) \in \mathbb{G}_T$$

Decryption: Let $C = [U_0, U_2, \cdots, U_t, V]$ be the ciphertext encrypted using the ID-tuple $(\mathsf{ID}_1, \cdots, \mathsf{ID}_t)$. To decrypt C, E_t computes:

$$V \oplus H_2(\frac{\hat{e}(U_0, S_t)}{\prod_{i=2}^{t} \hat{e}(Q_{i-1}, U_i)}) = M$$

Boneh *et al.*'s HIBE [20]

In the above described HIBE scheme, the size of the ciphertext is linearly proportional to the size of the identity used in the encryption. The increase in the ciphertext size also results in an increase in the decryption cost. Therefore, Boneh *et al.* [20] proposed a new HIBE scheme with constant ciphertext size and decryption cost. In this scheme, the public keys (identities ID) at depth k are vectors of elements in $(Z_p^*)^k$, and $\mathsf{ID} = (I_1, \cdots, I_k) \in (Z_p^*)^k$. The j-th component corresponds to the identity at level j. We describe the algorithms as follows:

Setup(l): First, let $\mathbb{G}_1$ be a bilinear group of prime order p and let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ be a bilinear map. To generate system parameters for a HIBE of maximum depth l, select a random generator $g \in G_1$, a random $\alpha \in Z_p$, and set $g_1 = g^\alpha$. Next, pick random elements $g_2, g_3, h_1, \cdots, h_l \in G_1$. The message space for the scheme is $\mathcal{M} \in \mathbb{G}_T$. The public parameters and the master secret key are

$$\mathsf{pk} = (g, g_1, g_2, g_3, h_1, \cdots, h_l), \qquad \mathsf{mk} = g_2^\alpha$$

KeyGen($d_{\mathsf{ID}|k-1}, \mathsf{ID}$): To generate a private key d_{ID} for an identity $\mathsf{ID} = (I_1, \cdots, I_k) \in (\mathbb{Z}_p^*)^k$ of depth $k \leq l$, using the master secret key, pick a random $r \in \mathbb{Z}_p$ and output

$$d_{\mathsf{ID}} = (g_2^{\alpha} \cdot (h_1^{I_1} \cdots h_k^{I_k} \cdot g_3)^r, \quad g^r, \quad h_{k+1}^r, \quad \ldots, \quad h_l^r) \in \mathbb{G}^{2+l-k}$$

The private key for ID can be generated incrementally, given a private key for the parent identity $\mathsf{ID}_{k-1} = (I_1, \ldots, I_{k-1}) \in (\mathbb{Z}_p^*)^{k-1}$, as required. Let

$$d_{\mathsf{ID}|k-1} = (g_2^{\alpha} \cdot (h_1^{I_1} \cdots h_{k-1}^{I_{k-1}} \cdot g_3)^{r'}, g^{r'}, h_k^{r'}, \ldots, h_l^{r'}) = (a_0, a_1, b_k, \ldots, b_l)$$

be the private key for ID_{k-1}. To generate d_{ID}, pick a random $t \in \mathbb{Z}_p$ and output

$$d_{\mathsf{ID}} = (a_0 \cdot b_k^{I_k} \cdot (h_1^{I_1} \cdots h_k^{I_k} \cdot g_3)^t, a_1 \cdot g^t, b_{k+1} \cdot h_{k+1}^t, \ldots, b_l \cdot h_l^t)$$

This private key is a properly distributed private key for $\mathsf{ID} = (I_1, \ldots, I_k)$ for $r = r' + t \in \mathbb{Z}_p$.

Encrypt$(params, ID, M)$: To encrypt a message $M \in G_T$ under the public key $\mathsf{ID} = (I_1, \ldots, I_k) \in (\mathbb{Z}_p^*)^k$, pick a random $s \in \mathbb{Z}_p$ and output

$$\mathsf{CT} = (\hat{e}(g_1, g_2)^s \cdot M, \quad g^s, \quad (h_1^{I_1} \cdots h_k^{I_k} \cdot g_3)^s) \in \mathbb{G}_T \times \mathbb{G}_1^2$$

Decrypt$(d_{\mathsf{ID}}, \mathsf{CT})$: Consider an identity $\mathsf{ID} = (I_1, \ldots, I_k)$. To decrypt a given ciphertext $\mathsf{CT} = (A, B, C)$ using the private key $d_{\mathsf{ID}} = (a_0, a_1, b_{k+1}, \ldots, b_l)$, output

$$A \cdot \hat{e}(a_1, C) / \hat{e}(B, a_0) = M$$

Subsequently, several HIBE schemes are proposed in the literature. Boyen and Waters introduced an anonymous HIBE scheme in [29] where the ciphertext of their HIBE scheme does not leak any information about the identities to which the messages are encrypted. An anonymous HIBE scheme can be extended to applications such as keywords searchable encryption where keywords can be in a hierarchical structure.

2.3.3 Role-based Encryption

Several role-based encryption (RBE) schemes [148, 150, 149] have been proposed that specifically addresses the security of outsourced data with RBAC policies. We describe the scheme [150] as follows.

Zhu *et al.*'s Role-based Encryption [150]

Let $\{U, R, \preceq\}$ is a role-key hierarchy with partial-order $\preceq$, and U is the set of the users $\{u_1, u_2, \cdots, u_n\}$ and R is the set of roles $\{r_1, r_2, \cdots, r_m\}$.

Setup(s, λ) Generate three groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, and a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, and choose random generators $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$. Choose a random $\tau_0 \in \mathbb{Z}_p^*$ and a random integer $\tau_i \in \mathbb{Z}_p^*$ for each role r_i in role key hierarchy, and computes

$$D_i = g^{\tau_i} \ \in \ \mathbb{G}_i, \forall r_i \in R, \quad V = e(g, h) \ \in \ \mathbb{G}_T$$

Each τ_i and U_i are the secret and the identity of the role r_i respectively. The public parameter is

$$params = \{h, V, D_0, D_1, \cdots, D_m\}, \text{ where } D_0 = g^{\tau_0}$$

and the master secret is

$$mk = \{g, \tau_0, \tau_1, \cdots, \tau_m\}$$

GenRKey($params$, r_i). This algorithm assigns the role encryption key by computing the follows for each role r_i.

$$pk_i = \{H, V, W_i, \{D_k\}_{r_k \in \uparrow r_i}\} \quad \text{where} \ \ W_i = D_0 + \sum_{r_i \not\preceq r_k} D_k = g^{\zeta_i}$$

where $\{U_k\}_{r_k \in \uparrow r_i}$ is the set of all the ancestor roles denoted $\uparrow r_i$, which denotes the control domain for the role r_i. Note that $W_i = g^{\tau_0 + \sum_{r_i \not\preceq r_k} \tau_k}$, so there is

$\zeta_i = \tau_0 + \sum_{r_i \npreceq r_k} \tau_k$.

AddUser($mk, ID, u_{i,j}$). Given $mk = \{g, \tau_0, \cdots, \tau_m\}$ and a user index $u_{i,j}$ in the role r_i, the manager generates a unique decryption key by randomly selecting a $x_{i,j} = Hash(ID, u_{i,j}) \in \mathbb{Z}_p^*$, and set $x'_{i,j} = x_{i,j} - \sum_{r_i \npreceq r_k} \tau_k \in \mathbb{Z}_p^*$, and then defining a public user label $lab_{i,j} = \langle x_{i,j}, A_{i,j}, B_{i,j} \rangle$ and the user decryption key $dk_{i,j} = A_{i,j}$ where

$$A_{i,j} = g^{x_{i,j}/(\zeta_i + x'_{i,j})} \in \mathbb{G}_1, \quad B_{i,j} = h^{1/(\zeta_i + x'_{i,j})} \in \mathbb{G}_2, \quad V_{i,j} = V^{1/(\zeta_i + x'_{i,j})} \in \mathbb{G}_T$$

Encrypt($\mathcal{R}, pk_i, M$). To encrypt the message $M \in \mathbb{G}_T$, given any $pk_i = \{H, V, W_i, \{D_k\}_{r_k \in \uparrow r_i}\}$ and a set of revoked users $\mathcal{R} = \{u_{i_1,j_1}, \ldots, u_{i_t,j_t}\}$, the algorithm randomly picks $t \in \mathbb{Z}_p^*$ and then computes

$$C_1 = W_i^t, \quad C_2 = (B_{\mathcal{R}})^t, \quad C_3 = M \cdot (V_{\mathcal{R}})^t, \quad C_4 = \{(D_k)^t\}_{\forall r_k \in \uparrow r_i}$$

and outputs ciphertext as $C = (C_1, C_2, C_3, C_4, \mathcal{R})$.

In the above algorithm, $B_{\mathcal{R}}$ and $V_{\mathcal{R}}$ are computed as

$$B_{\mathcal{R}} = \begin{cases} h^{1/\prod_{l=1}^{t}(\zeta_{i_l} + x'_{i_l,j_l})}, & \text{if } \mathcal{R} \neq \varnothing \\ h, \quad \text{if } \mathcal{R} = \varnothing \end{cases}$$

$$V_{\mathcal{R}} = \begin{cases} V^{1/\prod_{l=1}^{t}(\zeta_{i_l} + x'_{i_l,j_l})}, & \text{if } \mathcal{R} \neq \varnothing \\ V, \quad \text{if } \mathcal{R} = \varnothing \end{cases}$$

Decrypt($dk_{i,j}, C$). Given a ciphertext C from the role r_i, the k-th user $u_{j,k}$ in the role r_j can recover the message M with $dk_{j,k} = A_{j,k}$, when $r_i \preceq r_j$ and $u_{j,k} \notin \mathcal{R}$, by computing

$$V' = e(C_1 + \sum_{r_l \in \Gamma(r_j, r_i)} D'_l,\ B^{\mathcal{R}}_{j,k}) \cdot e(A_{j,k}, C_2)$$

where $\Gamma(r_j, r_i)$ denotes $\cup_{r_i \preceq r_k \preceq r_j \{r_k\}}$, $D'_l = (D_l)^t \in C_4$ for all $r_l \in \Gamma(r_j, r_i)$, and

$$B^{\mathcal{R}}_{j,k} = \begin{cases} h^{1/[\ \prod_{l=1}^{t}(\zeta_{i_l}+x'_{i_l,j_l})\cdot(\zeta_j+x'_{j,k})\ ]}, & \text{if } \mathcal{R} \neq \varnothing \\ B_{j,k}, \quad \text{if } \mathcal{R} = \varnothing \end{cases}$$

from $\{B_{i_l,j_l}\}_{u_{i_l,j_l}\in\mathcal{R}}$ and $B_{j,k}$. Then the user can output the message $M = C_3/V'$.

This scheme gives a cryptographic solution which can enforce RBAC policies in the encryption of data. The security of this encryption scheme is based on the security of the cryptographic algorithm. More specifically, when a user is assigned to a role in this encryption scheme, a key is calculated through a cryptographic algorithm by taking as input the master secret and identity of the user and role.

2.3.4 Attribute-based Encryption

Attribute-based encryption schemes can also be used to enforce role-based access policies which we will discuss later. Generally, there are two types of ABE schemes, KP-ABE and CP-ABE. KP-ABE schemes cannot be used to enforce RBAC due to the feature that the owner of the data does not have the control over who is allowed to access the data. Now let us see how to enforce the RBAC policies using CP-ABE schemes.

First we associate the role with a policy of a set of attributes, and we say that a user belongs to the role if this user has the keys for all the attributes in the set for the role. To define a role A which inherits the permissions from another role B, we simply associate role A with a policy which contains the policy for role B. When the owner wants to encrypt a message to a role, she or he simply uses the CP-ABE scheme to encrypt the message under the policy of the role, and all the users in the role will be able to decrypt as they have the keys for all the attributes in the policy. The ABE scheme that we used in comparison refers to this approach.

Goyal *et al.*'s KP-ABE scheme [63]

Goyal *et al.* [63] proposed the first ABE scheme, in which ciphertexts are labelled with sets of attributes and private keys are associated with access structures that control which ciphertexts a user is able to decrypt. Hence this scheme is also referred to as key-policy ABE or KP-ABE. The scheme has the following algorithms.

Setup Generate two groups $\mathbb{G}_1, \mathbb{G}_2$, and a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. Choose a random generator $g \in \mathbb{G}_1$, and a random $u \in \mathbb{Z}_p$. The Lagrange coefficient $\Delta_{i,S}$ is defined for $i \in \mathbb{Z}_p^*$ and a set S of elements in $\mathbb{Z}_p$: $\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x-j}{i-j}$.

Define a list of attributes $\mathcal{U} = \{1, 2, \cdots, n\}$. For each attribute $i \in \mathcal{U}$, choose a number t_i uniformly at random from $\mathbb{Z}_p$. The public parameters are

$$PK = (\ T_1 = g^{t_1}, \cdots, T_n = g^{t_n}, Y = \hat{e}(g, g)^y\ )$$

and the master secret is

$$MK = (t_1, \cdots, t_n, y)$$

Encryption(M, γ, PK). To encrypt a message $M \in \mathbb{G}_2$ under a set of attributes γ, choose a random value $s \in \mathbb{Z}_p$ and compute the ciphertext as,

$$C = (\gamma, C' = MY^s, \{C_i = T_i^s\}_{i \in \gamma}).$$

Key Generation(T, MK). The algorithm outputs a key that enables the user to decrypt a message encrypted under a set of attributes γ if and only if $T(\gamma) = 1$. The algorithm proceeds as follows. First choose a polynomial q_x for each node x including the leaves in the tree T. These polynomials are chosen in the following way in a top-down manner, starting from the root node r.

For each node x in the tree, set the degree d_x of the polynomial q_x to be one less than the threshold value k_x of that node, that is, $d_x = k_x - 1$. For the root node r, set $q_r(0) = y$ and d_r other points of the polynomial q_r randomly to define it completely. For any other node x, set $q_x(0) = q_{parent(x)}(index(x))$ and choose d_x other points randomly to completely define q_x.

Once the polynomials have been decided, for each leaf node x, give the following secret value to the user:

$$D_x = g^{\frac{q_x(0)}{t_i}}, \quad \text{where } i = attr(x).$$

The set of above secret values is the decryption key D.

Decrypt(C, D). This algorithm is defined in a recursive way. A recursive algorithm DecryptNode(C, D, x) is defined to take as input the ciphertext $C = (\gamma, C' = MY^s, \{C_i = T_i^s\}_{i\in\gamma})$, the private key D, and a node x in the tree. It outputs a group element of $\mathbb{G}_2$ or $\perp$.

Let $i = att(x)$. If the node x is a leaf node then:

$$\text{DecryptNode}(C, D, x) = \begin{cases} \hat{e}(D_x, C_i) = \hat{e}(g^{\frac{q_x(0)}{t_i}}, g^{s\cdot t_i}) & \text{if } i \in \gamma \\ \perp \quad \text{otherwise} \end{cases}$$

Considering the recursive case when x is a non-leaf node, DecryptNode(C, D, x) proceeds as follows: For all nodes z that are children of x, it calls DecryptNode and stores the output as F_z. Let S_x be an arbitrary k_x-sized set of child nodes z such that $F_z \neq \perp$. If no such set exists then the node was not satisfied and the function returns $\perp$. Otherwise, denote $i = index(z), S'_x = \{index(z) : z \in S_x\}$, and computes:

$$\begin{aligned} F_x &= \prod_{z\in S_x} F_z^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z\in S_x} (\hat{e}(g, g)^{s\cdot q_z(0)})^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z\in S_x} (\hat{e}(g, g)^{s\cdot q_{parent(z)}(index(z))})^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z\in S_x} \hat{e}(g, g)^{s\cdot q_x(i)\cdot\Delta_{i,S'_x}(0)} \\ &= \hat{e}(g, g)^{s\cdot q_x(0)} \end{aligned}$$

and return the result.

Since DecryptNode$(C, D, r) = \hat{e}(g, g)^{ys} = Y^s$ if and only if the ciphertext satisfies the tree, and $C' = MY^s$, the decryption algorithm simply divides out Y^s and recovers the message M.

Bethencourt *et al.*'s CP-ABE scheme [12]

Another form of the ABE scheme was introduced by Bethencourt *et al.* [12]. This scheme works in the reverse manner where the user keys are associated with sets of attributes and the ciphertexts are associated with the policies. Hence it is referred to as the ciphertext-policy ABE (CP-ABE) scheme. The scheme is described as follows:

Setup Generate two groups $\mathbb{G}_1, \mathbb{G}_2$, and an bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. Choose a random generator $g \in \mathbb{G}_1$, and two random exponents $\alpha, \beta \in \mathbb{Z}_p$, and a hash function $H : \{0,1\}^* \rightarrow \mathbb{G}_1$. The Lagrange coefficient $\Delta_{i,S}$ is defined for $i \in \mathbb{Z}_p^*$ and a set S of elements in $\mathbb{Z}_p$: $\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x-j}{i-j}$.

Define a list of attributes $\mathcal{U} = \{1, 2, \cdots, n\}$. The public parameter is

$$PK = (\ \mathbb{G}_1, g, h = g^\beta, f = g^{1/\beta}, \hat{e}(g,g)^\alpha\ )$$

and the master secret is

$$MK = (\beta, g^\alpha)$$

Encryption(M, T, PK). The encryption algorithm encrypts a message M under the tree access structure T. The algorithm first chooses a polynomial q_x for each node x including the leaves in the tree T. These polynomials are chosen in the following way in a top-down manner, starting from the root node r. For each node x in the tree, set the degree d_x of the polynomial q_x to be one less than the threshold value k_x of that node, that is, $d_x = k_x - 1$.

For the root node r, set $q_r(0) = s$ and d_r other points of the polynomial q_r randomly to define it completely. For other nodes x, set $q_x(0) = q_{parent(x)}(index(x))$ and choose d_x other points randomly to completely define q_x.

Let Y be the set of leaf nodes in T. The ciphertext is then constructed by giving the tree access structure T and computing the ciphertext CT as

$$T, C' = M\hat{e}(g,g)^{\alpha s}, C = h^s$$

$$\{C_y = g^{q_y(0)}, C'_y = H(att(y))^{q_y(0)}\}_{y \in Y}.$$

Key Generation(MK, S). The algorithm takes as input a set of attributes S and outputs a key that identifies with that set. First choose a random $r \in \mathbb{Z}_p$, and then pick random $r_j \in \mathbb{Z}_p$ for each attribute $j \in S$. Then it computes the key as

$$SK = (D = g^{(\alpha+r)/\beta}, \{D_j = g^r \cdot H(j)^{r_j}, D'_j = g^{r_j}\}_{j \in S}).$$

Decrypt(CT, SK). First, an recursive algorithm DecryptNode(CT, SK, x) is defined to take as input the ciphertext $CT = (T, C', C, \{C_y, C'_y\}_{y \in Y})$, the private key SK which is associated with a set S of attributes, and a node x in the tree.

Let $i = att(x)$. If the node x is a leaf node then:

$$\text{DecryptNode}(CT, SK, x) = \frac{\hat{e}(D_i, C_x)}{\hat{e}(D'_i, C'_x)} = \frac{\hat{e}(g^r \cdot H(i)^{r_i}, g^{q_x(0)})}{\hat{e}(g^{r_i}, H(i)^{q_x(0)})} = \hat{e}(g,g)^{r \cdot q_x(0)}$$

If $i \notin S$, then define DecryptNode(CT, SK, x) $=\perp$.

Considering the recursive case when x is a non-leaf node, DecryptNode(CT, SK, x) proceeds as follows: For all nodes z that are children of x, it calls DecryptNode and stores the output as F_z. Let S_x be an arbitrary k_x-sized set of child nodes z such that $F_z \neq \perp$. If no such set exists then the node was not satisfied and the function returns $\perp$.

Otherwise, having $i = index(z), S'_x = \{index(z) : z \in S_x\}$, compute:

$$\begin{aligned} F_x &= \prod_{z \in S_x} F_z^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z \in S_x} (\hat{e}(g,g)^{r \cdot q_z(0)})^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z \in S_x} (\hat{e}(g,g)^{r \cdot q_{parent(z)}(index(z))})^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z \in S_x} \hat{e}(g,g)^{r \cdot q_x(i) \cdot \Delta_{i,S'_x}(0)} \\ &= \hat{e}(g,g)^{r \cdot q_x(0)} \end{aligned}$$

and return the result.

Then the decryption algorithm begins by simply calling the function on the root node r of the tree T. If the tree is satisfies by S, set $A = \text{DecryptNode}(CT, SK, r) = \hat{e}(g,g)^{rq_r(0)} = \hat{e}(g,g)^{rs}$. The algorithm now decrypts by computing

$$M = C'/(\hat{e}(C,D)/A) = C'/(\hat{e}(h^s, g^{(\alpha+r)/\beta})/\hat{e}(g,g)^{rs}).$$

We have shown that an ABE scheme can be used to enforce RBAC policies. However, in that approach, the size of users' decryption 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. There are also several other approaches which aimed to implement cryptographic RBAC using ABE schemes. Ferrara *et al.* [55] defined a special case of predicate encryption [86], called predicate encryption for non-disjoint sets (PE-NDS), using generic ABE schemes and showed how to use a PE-NDS scheme to implement a cryptographic RBAC scheme using a similar approach as what we described above. The user revocation, however, in their scheme is performed by adding a new attribute to the role. Hence, in the case of user revocation, all the files encrypted to the role need to be re-encrypted to prevent revoked users from accessing previously encrypted files. Zhu *et al.* [149] investigated the solutions of using an ABE scheme in a 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. In this thesis, we show how the weaknesses and the missing aspects of using ABE schemes can be addressed by our role-based encryption schemes.

2.4 Trust Model

2.4.1 Experience-based Trust

Trust has played a foundational role in security for a long period of time. Perhaps the most notable one was in the development of Trusted Computer System Evaluation Criteria (TCSEC) [47] in the late 70s and early 80s, where trust was used in the process of convincing the observers that a system (model, design or implementation) was correct and secure. This led to the notions of trusted processes in operating systems and the Trusted Computing Base (TCB). In the mid 1980s and 1990s, trust played a key role in the context of distributed system security, leading to the formulation of trusted authorities involved in the management of security services. Several trusted authorities such as authentication and access control authorities were introduced. Such authorities, which are still in use, are "trusted" by the population of entities that use them to provide guarantees and assurances about the security information and services that they provide. In the mid 1990s, the idea of trust management [15, 16] became popular. It was originally proposed to primarily address the management of identity and authentication information, and then was expanded to authorisation and privilege information. In the late 1990s and early 2000s, another significant development in the security space was the notion of a trusted platform. This notion of a trusted platform was introduced by the Trusted Computing Platform Alliance (TCPA), currently known as TCG. A trusted platform is one that contains a hardware-based subsystem devoted to maintaining trust and security between machines. It has some "special processes" which dynamically collect and provide evidence of behaviour. These special processes themselves are "trusted" to collect evidence properly. There are also trusted third parties endorsing platforms which underlie the confidence that the platform can be "trusted".

It is clear that two entities normally do not trust each other on the identity alone. There are a range of other attributes and credentials, such as different types of privileges, the state of the platform being used as well as reputations, recommendations and histories that come into play in decision making. Most importantly, such information can be dynamic and can vary in different ways in different contexts. We refer to those beliefs that are derived from concrete security mechanisms and attributes such as certificates and credentials as "hard trust". They are often characterised by "certainty". On the other hand, beliefs derived from social control mechanisms and intangible information such as reputation and experiences are referred to as "soft trust". These beliefs are characterised by "uncertainty", dependent on past behaviour and often involve recommendations from multiple entities and they are progressively tuned over time. Recently, a number of models have been developed using soft trust techniques to determine the trustworthiness of systems [50, 134, 84]. The experience-based trust model is one such trust management system which enables the trust decisions to be made based on the historical behaviour of an entity [81, 105]. Such a system allows an entity to rate the transactions with other entities, and the trustworthiness of an entity is determined using the collection of ratings of the transactions that other entities have had with this entity.

Most experience-based trust systems derive the trustworthiness of an entity from both its own experience and the feedback on the transactions provided by other entities which have had interactions with the entity concerned in the past. Let us consider a simple example of such a system. When a client c finishes a transaction with a service provider p, c gives a feedback as either "positive" or "negative" depending on whether or not it is satisfied with the transaction. The feedback record is of the form $f = (c, p, b, t)$ where b represents the binary value of the feedback and t is the timestamp when the transaction took place. This record f is uploaded by the client to a trust central repository. When another client wants to evaluate the trustworthiness of the service provider p (assuming this client does not have any previous experience with the provider), first it obtains the collection of feedback records $Hist(p) = \{f_1, \ldots, f_n\}$

from the central repository, where n is the total number of the feedbacks about p that have been uploaded to the central repository; $Hist(p)$ represents the feedbacks that all the clients have made to the service provider p. By adding up the total number of each different type of feedback, the client gets an evidence tuple (p, r, s) where r represents the total number of "positive" feedbacks appeared in the collection $Hist(p)$, and s is the total number of "negative" feedbacks in $Hist(p)$. Then the client makes the decision whether or not to continue the transaction with the service provider p based on whether this tuple exceeds a certain threshold; this threshold is dependent on the context of the application at hand.

2.4.2 Average of Ratings

The simplest and the most straightforward approach to compute trust scores based on historical feedback is that the total score is the difference between the positive score and the negative score which are calculated as the total number of positive ratings and negative ratings separately. This approach intuitively shows an entity's trustworthiness derived from its past behaviour, and it is easy to understand and simple to implement. This is the approach used in eBay's reputation forum, and it has been described in detail in [112].

In this trust model, the historical feedback is the only factor considered in calculating trust scores, and the trust model is fair under the assumption that all the ratings are precise and accurate. However, this is unlikely to be true in almost every system as human opinions are usually quite subjective and it is very rare that different persons would rate an entity based on exactly the same standard. An improvement can be made by applying a weight to the average of all the ratings, such as the approach in [122], where the rating weight can be determined by factors such as trustworthiness or reputation of the entity which provides the ratings, current trust scores of the target entity, etc.

2.4.3 Bayesian Trust Models

Many approaches have been proposed that use probabilistic models to evaluate the trust based on the evidence tuple which contains the number of "positive" and "negative" transactions in which the given entity has been involved. Perhaps the most common probabilistic model is the one based on Bayesian trust using a beta probability distribution function[105, 83, 104]. The beta family of distributions is a collection of continuous probability density functions defined over the interval [0, 1]. Suppose a beta distribution used for a parameter θ is defined as

$$P(\theta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}$$

where α and β are two parameters controlling the distribution of the parameter θ, and $0 \leq \theta \leq 1,\ \alpha > 0,\ \beta > 0$. Assume $X = \{x_1, \ldots, x_n\}$ is the collection of the feedback from the past n transactions, and X has r "positive" feedbacks and s "negative" feedbacks. Then the likelihood function can be defined as

$$P(X|\theta) = \prod_{i=1}^{n} P(x_i|\theta) = \theta^r(1-\theta)^s$$

The posterior distribution $P(\theta|X)$ is proportional to the multiplication of the prior $P(\theta)$ and the likelihood function $P(X|\theta)$, and we then have

$$\begin{aligned} P(\theta|X) &= \frac{P(X|\theta)P(\theta)}{P(X)} \\ &= \frac{\Gamma(r+\alpha+s+\beta)}{\Gamma(r+\alpha)\Gamma(s+\beta)}\theta^{r+\alpha-1}(1-\theta)^{s+\beta-1} \end{aligned}$$

Now let x_{i+1} be the possible feedback of the next transaction. The probability that x_{i+1} is a "positive" feedback given the transaction history X can be represented as

$$\begin{aligned} P(x_{i+1}|X) &= \int_0^1 d\theta\ P(x_{i+1}|\theta)P(\theta|X) \\ &= \int_0^1 d\theta\ \theta P(\theta|X) \\ &= E(\theta|X) \end{aligned}$$

Then we write the probability that the next transaction will be a “good” one as follows:

$$\mathcal{E}(r,s) = P(x_{i+1}|X) = E(\theta|X) = \frac{r+\alpha}{r+\alpha+s+\beta} \tag{2.1}$$

Using Equation 2.1, the client can derive the probability that the next transaction with the provider will be positive from the transaction history of the provider. Most Bayesian trust systems assume that the parameters $\alpha = \beta = 1$. Some other approaches allows the parameters α and β to be chosen depending on the system context.

2.4.4 Belief Trust Models

So far in the described trust models, the trust ratings are either positive or negative. The belief trust model is a type of probabilistic model where a distinct probability interpreted as “uncertainty” is used to represent the trustworthiness of entities. A trust metric for uncertain probabilities, called opinion, was described by Josang [82, 81]. An entity's opinion about a statement x represents the entity's belief in the truth of the statement x, and it is denoted by a set of four elements as follows,

$$\omega_x = (b_x,\ d_x,\ u_x,\ a_x)$$

where b_x, d_x, and u_x represent the probability of belief, disbelief and uncertainty respectively, and $a_x \in [0,1]$, called the relative atomicity, represents the base rate probability in the absence of evidence. The belief of an entity can only be one of these three types; that is, $b_x + d_x + u_x = 1$.

An opinion's probability expectation value which represents the probability that the next transaction will be a “good” one is defined as

$$\mathcal{E}(\omega_x) = b_x + a_x u_x$$

A set of logical operators are described and used for logical reasoning with uncertain propositions. A *discounting* operator is used (when an entity A holds an opinion about another entity B) to derive A's opinion about a proposition x from B's opinion about x. A *consensus* operator is used (when two entities hold separate opinions about a proposition x) to combine two opinions into one which can reduce the uncertainty. Josang [81] shows that opinions can be uniquely mapped to beta PDFs, and the consensus operator in the belief trust model is equivalent to the updating operations in the Bayesian model. Yu and Singh [135] have proposed an approach to use the belief model to represent reputation scores.

2.5 Conclusion

In this chapter, we have briefly described several well-known access control models and the cryptographic approaches which can be used to enforce access policies of different access control models in an untrusted environment. Then we discussed the cryptographic RBAC schemes, and showed how the existing cryptographic schemes can be used to enforce RBAC policies. We have discussed the weaknesses of these cryptographic RBAC solutions. Lastly, we have reviewed the concept of trust models, and described several commonly used probabilistic models.

3

Generic Role-based Encryption Frameworks

In this chapter, we define the formulation of role-based encryption (RBE) and propose three generic constructions which make use of ID-based broadcast encryption (IBBE) techniques to build RBE schemes. Our constructions show that IBBE techniques can be used in different ways to build RBE schemes with different features. The constructed RBE schemes allow the owner of data to specify a set of roles to which she or he wishes to grant permission for accessing the data, and encrypt the data in such a way that only the users in these roles can decrypt and view the plain data. In addition, we show that these constructions are able to deal with role hierarchies, whereby roles can inherit permissions from other roles. Moreover, these generic constructions are able to deal efficiently with the situation where a user belongs to multiple roles, and the size of the user's decryption keys remains constant regardless of the number of roles that the user has been assigned to.

Another advantage is that our RBE constructions are generic; that is, any secure IBBE scheme can be converted into a RBE scheme. When developing a RBE scheme, designers may require their scheme to be built using certain cryptographic technique such as bilinear pairing or ideal lattice, possibly because they want the scheme to be more efficient or more secure or even easier to implement. Though it is possible to propose a specific RBE scheme directly using a particular cryptographic tool, such an approach requires designing a new scheme each time and proving its security. Therefore, generic constructions which can work with any secure IBBE schemes provide a convenient solution for designing a RBE scheme; a suitable IBBE scheme can be chosen for the RBE scheme, balancing security and performance characteristics depending on the system requirements.

This chapter is organised as follows. In section 3.1 we give the definition of RBE schemes. Section 3.2 defines the security properties of RBE schemes. In section 3.3, we describe the security requirements for the IBBE schemes used in the generic RBE constructions. Section 3.4 describes our generic constructions for RBE schemes. We provide a security analysis of our constructions in section 3.5. In section 3.6, we illustrate and compare the features of our generic RBE constructions using examples. Finally, section 3.7 concludes the chapter.

3.1 Formulation of Role-based Encryption

Although there are many approaches in the literature, which can enforce RBAC policies on outsourced data, there is no standard definition for such a scheme that can meet all the security requirements discussed in section 1.2. In this section, we give the formal definition of a RBE scheme and describe the participants involved in a RBE scheme.

We first define four types of entities which are involved in a RBE scheme.

- SA, the system administrator of the system. It generates the system parameters and issues all the necessary credentials. In addition, this administrator manages the role hierarchy structure for the RBAC system.

- RM is a role manager who manages the user membership of a role. In systems where there are a small number of users, the SA can act as the role manager to manage the user membership of each role to keep the systems compact. However, in large-scale systems, it is almost impractical for a single party to manage all the users and permissions. Therefore, having separate role managers can make the user management tasks more flexible and efficient.

- **Users** are the parties who want to access and decrypt the stored data. Each user is required to be authenticated by the SA and issued a credential, which is associated with the identity of the user, upon successful authentication.

- **Owners** are the parties who possess data and want to store the encrypted data in the cloud for other users to access. Owners define role-based access policies to specify who can access their data. An owner can either be a user within the system or an external party who wants to send data to users in the system. In this thesis, we consider an owner to be a logically separate component even though a user can be an owner and vice versa.

We then define the following algorithms for the 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 $\mathcal{U}$ which will list all the users who are the members of that role is also returned to the RM.

ManageRole (mk, sk_R, ID_R, $\mathcal{T}$) is executed by the SA to manage a role with the identity ID_R in the role hierarchy $\mathcal{T}$. This operation returns a set of public parameters pub_R to the role.

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

RevokeUser (pk, sk_R, $\mathcal{U}_R$, 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 $\mathcal{U}_R$, being updated.

Encrypt (pk, $\mathcal{R}$, M) is executed by the owner of a message M. This algorithm takes as input the system public key pk, the role public parameters of the set of roles $\mathcal{R}$, and outputs the ciphertext C.

The algorithm employs a secure symmetric encryption scheme *Enc* to perform the encryption of the message M, and there are two phases in this algorithm. The first phase selects a random secret key K of the symmetric encryption scheme *Enc* and encrypts the key K using the public parameters. Then the second phase uses the encryption scheme *Enc* to encrypt the message M with the secret key K. The output ciphertext C is comprised of the ciphertext of both the key K and the message M. In this thesis, when we describe RBE constructions, we only describe the first phase of this algorithm for the simplicity purpose. When we use the input M, we are referring to the symmetric encryption key K, and we assume that a secure symmetric encryption scheme will be used to generate the ciphertext of the real message M.

Decrypt (pk, 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 of the role R, the user decryption key dk, the ciphertext C, and outputs the message M.

We assume that the cloud provider provides an underlying service to store the system public parameters and all the role public information. The SA is a trusted party who creates the roles and the keys for the users and the roles. We assume that the master secret key mk is securely stored by the SA, and the SA distributes the secret information, such as keys, to the role manager RM and users via a secure channel.

The role manager (RM) is a trusted party per role and it manages the set of users in a given role. It can assign a role to a user if the user qualifies for the role, or exclude a user if the user is found to be malicious. When adding a user to a role, RM needs to verify the user's qualifications and determine if the user can have this role. When a user leaves a role or is removed from a role, RM revokes the role permissions for that user. We assume that there are some standard authentication mechanisms available that can be performed between a RM and a user.

Users are given appropriate decryption keys by the SA when they join the system to access ciphertexts stored in the cloud. The users possess some credentials which are used to prove to an RM that they have appropriate qualifications to join a role. Users are able to use their decryption keys to decrypt the ciphertext, and we assume that the users are responsible for keeping their decryption keys secure.

3.2 Security Properties of Role-based Encryption

In this section, we define the security properties of a RBE scheme. We build the security of the scheme on the standard notion of selective-ID security. In a RBE scheme, users who have never been granted membership of a role as well as the users who have been granted membership that has been later revoked 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 of the RBE scheme. It outputs the system public key pk to the $\mathcal{A}$, and keeps the master secret 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 types:

- *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 added as a member 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. We allow ID_R to be one of the identities that appears in the query in Phase 1. However, if the identity ID_R has been queried in Phase 1, the challenger checks if the identity of any existing member of the role ID_R was added to the role in the *AddUser* query in Phase 1. If there are some, the challenger executes *RevokeUser* algorithm to exclude these identities from the role and updates pub_R to ensure that no existing member was added to the role R in the queries made by the adversary $\mathcal{A}$ in Phase 1. Then 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 $\mathcal{A}$' advantage in attacking the role-based encryption scheme as $\mathsf{Adv}^{\mathcal{RBE}} = |Pr[b = b'] - \frac{1}{2}|$ and we have the following definition.

Definition 3.1 *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 3.2 *We say that a role-based encryption scheme is CPA secure if it is $(\epsilon, n, q_E,\ q_A, 0)$ CCA secure.*

3.3 Preliminaries

3.3.1 Key-Dependent Message Security

In this section, we begin the discussion by reviewing the definition of Key-Dependent Message security (KDM). A KDM secure encryption (also referred to circular-secure encryption) scheme ensures that the adversary cannot distinguish the encryption of a key-dependent message from an encryption of 0. The KDM security has been discussed in many works, such as in [32, 14, 26, 31, 3, 2], in which [2] has given the definition of the KDM security in the context of ID-based encryption. In this chapter, we use the KDM security based on the definition given in [2].

Let $\mathcal{E} = (\mathsf{Setup}, \mathsf{Ext}, \mathsf{Enc}, \mathsf{Dec})$ be an ID-based encryption scheme. $\mathsf{Ext}(\mathsf{mk}, \mathsf{ID})$ is a key-generation algorithm that takes as input an identity ID and a master secret key mk and outputs a secret key sk. $\mathsf{Enc}(\mathsf{pk}, \mathsf{ID}, m)$ is the encryption algorithm that uses an identity ID and the public key pk to encrypt message m and outputs a ciphertext c, and $\mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, c)$ is the decryption algorithm that decrypts ciphertext c using private key sk and discovers message m. We denote the secret key space of $\mathcal{E}$ as $\mathcal{K}$, and the message space of $\mathcal{E}$ as $\mathcal{M}$.

We define a finite set $\mathcal{F} := \{f : \mathcal{K}^n \to \mathcal{M}\}$ of functions to represent the key dependencies, and we assume that the output size of these functions are independent of the input.

The KDM security is defined as a game between an adversary $\mathcal{A}$ and a challenger as below.

Init: The adversary $\mathcal{A}$ outputs 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 runs the algorithm Setup and generates a master secret key mk and a system public system pk. Then the challenger randomly chooses a bit $b \in \{0, 1\}$.

Queries:

- *Extract query:* The adversary $\mathcal{A}$ submits any identity $\mathsf{ID}_k \notin U$ to the challenger. The challenger executes the Ext algorithm on the identity ID_k and returns the generated key $\mathsf{sk}_i = \mathsf{Ext}(\mathsf{mk}, \mathsf{ID}_k)$ to $\mathcal{A}$.
- *Encryption query:* The adversary $\mathcal{A}$ adaptively issues queries in the form (i, f) where $i \in [1, n]$ and $f \in \mathcal{F}$. The challenger computes a message $S = f(\mathsf{sk}_1, \mathsf{sk}_2, \cdots, \mathsf{sk}_n)$, chooses an identity $\mathsf{ID}_i \in U$ and responds with $c = \mathsf{Enc}(\mathsf{pk}, \mathsf{ID}_i, S)$.

Challenge: When the adversary $\mathcal{A}$ decides that all the queries have been completed, it computes a message $M = f(\mathsf{sk}_1, \mathsf{sk}_2, \cdots, \mathsf{sk}_n)$, and sends to the challenger. The challenger chooses an identity $\mathsf{ID}_i \in U$ and responds with $c = \mathsf{Enc}(\mathsf{pk}, \mathsf{ID}_i, M)$ if $b = 0$, or $c = \mathsf{Enc}(\mathsf{pk}, \mathsf{ID}_i, 0^{|M|})$ if $b = 1$.

Guess: By using the responses from the challenger in the above phases, the adversary $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$, and wins the game if $b = b'$.

We denote the advantage that $\mathcal{A}$ wins the game as Adv^n. Then we have the following definition.

Definition 3.3 *A public key encryption scheme is selective-ID ε-KDM secure if for all polynomially bounded adversary $\mathcal{A}$ the advantage Adv^n that $\mathcal{A}$ wins the game is negligible. That is,*

$$\mathsf{Adv}^n = |Pr[b = b'] - \frac{1}{2}| < \varepsilon$$

3.3.2 ID-based Broadcast Encryption

In a RBE scheme, users can have their individual secret keys to decrypt data encrypted to the role to which they belong. This is analogous to broadcast encryption schemes where users can use their own private keys to decrypt messages encrypted by a broadcaster. Therefore, we use IBBE schemes to build our RBE constructions.

In order to build a secure RBE scheme, we require that the IBBE scheme to be used in the generic RBE constructions is KDM secure. We note that there is no specific IBBE scheme in the literature that was designed to be KDM secure. However, several generic KDM secure scheme conversions have been proposed [32, 14, 3] which can convert a non KDM secure encryption scheme to a KDM secure one.

We have described the algorithms of IBBE schemes in section 2.2.1. Now we show how to apply the generic conversion in [32] on an IBBE scheme to build a KDM secure IBBE scheme. First we define a hash function $H : \mathcal{M} \rightarrow \mathcal{M}^*$ where $\mathcal{M}$ is the message space of the original IBBE scheme. Assume that the $\mathcal{IBBE}'$ represents the original IBBE scheme in the description, we build the KDM secure IBBE scheme in the random oracle model as follows:

$\mathcal{IBBE}.\mathcal{S}etup(\lambda) := \mathcal{IBBE}'.\mathcal{S}etup(\lambda)$

$\mathcal{IBBE}.\mathcal{Extract}(\mathsf{mk},\ \mathsf{ID}) := \mathcal{IBBE}'.\mathcal{Extract}(\mathsf{mk},\ \mathsf{ID})$

$\mathcal{IBBE}.\mathcal{Encrypt}(\mathsf{pk}, \mathcal{U}, M)$: First choose a random $r \xleftarrow{R} \mathcal{M}$, and then output the ciphertext

$$C = (c_1, c_2) = (\mathcal{IBBE}'.\mathcal{Encrypt}(\mathsf{pk}, \mathcal{U}, r),\ H(r) \oplus M)$$

$\mathcal{IBBE}.\mathcal{Decrypt}(\mathsf{pk}, \mathsf{sk}, C) := H(\mathcal{IBBE}'.\mathcal{Decrypt}(\mathsf{pk}, \mathsf{sk}, c_1)) \oplus c_2$

Applying the above conversion on the IBBE scheme before using it in our generic RBE constructions will ensure that the constructed RBE scheme has the same level of security as the original IBBE scheme which we will prove later.

To use an IBBE scheme in our generic RBE constructions, we need a collision resistant hash function that maps user private keys to the message space of the IBBE scheme. However, in this chapter, we assume that the user private keys are encoded as elements of the message space to simplify the notation used in descriptions of our RBE constructions.

3.4 Generic RBE Constructions

In this section, we show three generic constructions to build RBE schemes using IBBE techniques. Each construction has its own advantages in terms of complexity of management and computational efficiency. To achieve revocation in these constructions, we assume that each role is assigned a set of pseudo-identities $\mathcal{ID}_R = \{\mathsf{ID}_R^1, \mathsf{ID}_R^2, \cdots, \mathsf{ID}_R^n\}$ as its role identity, and the real identity ID_R is only used to track the role and its associated parameters in the system. In this section, when we use the term role identity, we are referring to the pseudo-identities unless explicitly specified. The public parameters of each role consists of two parts, and we write them as $\mathsf{pub}_R = \{\mathsf{pub}_R^1, \mathsf{pub}_R^2\}$. The part pub_R^1 is used in specifying the role inheritance relationship within the role hierarchy, and the part pub_R^2 is used to define the user memberships of the role. They can be changed independently in RBE operations.

In the following descriptions, we let $\mathcal{PR}_R$ denote the set of roles which are the ancestor roles of a role R, and write the set of roles that directly inherit permissions from the role R as $\mathcal{PR}_R^D \subseteq \mathcal{PR}_R$.

3.4.1 Construction Type A

In hierarchical key management schemes, assume that a key K_1 is used to derive another key K_n and the set of keys $\{K_2, \cdots, K_{n-1}\}$ are the keys that lie on the path between K_1 and K_n in the hierarchy. To recover the key K_n using K_1, the key holders first derive the key K_2 of its direct descendant, and then one level down until they derive the key K_n. Similarly, a RBE scheme can be built by running a recursive algorithm in the decryption for each level of the hierarchical tree. We describe our first generic RBE construction referred as Construction Type A in Figure 3.1.

Setup(λ): Generate system parameters $\{\mathsf{mk}, \mathsf{pk}\} \longleftarrow \mathcal{IBBE}.\mathcal{S}\mathit{etup}(\lambda)$.

Extract(mk, ID): When $\mathsf{ID} = \mathsf{ID}_U$ is an identity of a user U, the user decryption key is generated as $\mathsf{dk}_U \longleftarrow \mathcal{IBBE}.\mathcal{E}\mathit{xtract}(\mathsf{mk}, \mathsf{ID}_U)$. When $\mathsf{ID} = \mathcal{ID}_R$ is a set of pseudo-identities of a role R, the role secret keys are generated as $\mathsf{sk}_R^i \longleftarrow \mathcal{IBBE}.\mathcal{E}\mathit{xtract}(\mathsf{mk}, \mathsf{ID}_R^i)$ where $\mathsf{ID}_R^i \in \mathcal{ID}_R$.

ManageRole(mk, sk_R, ID_R, $\mathcal{T}$): We denote the set of existing users of the role as $\mathcal{U}_R$.

To update the position of the role R in the role hierarchy, first an unused pseudo-identity ID_R and the corresponding secret key sk_R are chosen, and then the role public parameters $\{\mathsf{pub}_R^1, \mathsf{pub}_R^2\}$ are updated as $\mathsf{pub}_R^1 \longleftarrow \mathcal{IBBE}.\mathcal{E}\mathit{ncrypt}(\mathsf{pk}, \mathcal{PR}_R^D, \mathsf{sk}_R)$ and $\mathsf{pub}_R^2 \longleftarrow \mathcal{IBBE}.\mathcal{E}\mathit{ncrypt}(\mathsf{pk}, \mathcal{U}_R, \mathsf{sk}_R)$.

In addition, the direct descendant roles of the role R need to update their public parameters to have their permissions inherited by the new role. For each direct descendant role R_k whose set of direct ancestor roles was $\mathcal{PR}_{R_k}^D$, the public parameters are updated as $\mathsf{pub}_{R_k}^1 \longleftarrow \mathcal{IBBE}.\mathcal{E}\mathit{ncrypt}(\mathsf{pk}, \mathcal{PR}_{R_k}^D \cup \mathsf{ID}_R, \mathsf{sk}_{R_k})$.

If the role R has descendant roles before the hierarchy changes, each previous descendant role R_i needs to choose a new unused pseudo-identity ID_{R_i} and the corresponding secret key sk_{R_i}, and then updates the role public parameters $\mathsf{pub}^1_{R_i} \longleftarrow \mathcal{IBBE}.\mathcal{E}ncrypt(\mathsf{pk}, \mathcal{PR}^D_{R_i}, \mathsf{sk}_{R_i})$ and $\mathsf{pub}^2_{R_i} \longleftarrow \mathcal{IBBE}.\mathcal{E}ncrypt(\mathsf{pk}, \mathcal{U}_{R_i}, \mathsf{sk}_{R_i})$.

AddUser(pk, sk_R, $\mathcal{U}_R$, ID_U): To add a user U with identity ID_U into a role R which has a set $\mathcal{U}_R$ of existing members, the public parameters pub^2_R of the role R is updated as $\mathsf{pub}^2_R \longleftarrow \mathcal{IBBE}.\mathcal{E}ncrypt(\mathsf{pk}, \mathcal{U}_R \cup U, \mathsf{sk}_R)$.

RevokeUser(pk, sk_R, $\mathcal{U}_R$, ID_U): In this construction, revoking a user affects all the descendant roles of the role. We denote a role set $\mathcal{R} = \mathcal{SR}_R \cup R$ where $\mathcal{SR}_R$ is the set of all the descendant roles of the role R, and the directly ancestor role set of each role $R_i \in \mathcal{R}$ as $\mathcal{PR}^D_{R_i}$.

When revoking a user U with identity ID_U from the role R which has a set $\mathcal{U}_R$ of existing members, first the manager of each role $R_i \in \mathcal{R}$ needs to choose a new unused pseudo-identity ID_{R_i} and the corresponding secret key sk_{R_i}, and then update the role public parameters $\mathsf{pub}^1_{R_i}$ as $\mathsf{pub}^1_{R_i} \longleftarrow \mathcal{IBBE}.\mathcal{E}ncrypt(\mathsf{pk}, \mathcal{PR}^D_{R_i}, \mathsf{sk}_{R_i})$. Then R updates its public parameters pub^2_R as $\mathsf{pub}^2_R \longleftarrow \mathcal{IBBE}.\mathcal{E}ncrypt(\mathsf{pk}, \mathcal{U}_R \backslash U, \mathsf{sk}_R)$.

Encrypt(pk, $\mathcal{R}$, M): When encrypting the message M to a set of roles $\mathcal{R}$, the ciphertext is output as $C \longleftarrow \mathcal{IBBE}.\mathcal{E}ncrypt(\mathsf{pk}, \mathcal{R}, M)$.

Decrypt(pk, R, dk_U, C): Assume R is a role to which the message M is directly encrypted, R' is a role that inherits permissions from R. We denote the roles which are on the path between R' and R in the hierarchy as $\{R_1, \ldots, R_m\}$ where $R_1 = R'$ and $R_m = R$. To decrypt C, a user U of the role R' decrypts following the below algorithm.

```
set $\mathsf{sk} \longleftarrow \mathsf{dk}_U$
for $i = 1$ to $m$ do
```

```
    set $\mathsf{sk} \longleftarrow \mathit{IBBE.Decrypt}(\mathsf{pk}, \mathsf{sk}, \mathsf{pub}^1_{R_i})$
end for
output $M \longleftarrow \mathit{IBBE.Decrypt}(\mathsf{pk}, \mathsf{sk}, C)$
```

Figure 3.1: Generic RBE Construction Type A

In this construction, decryption has been split into separate levels. A user can decrypt the secret key of the role that she or he belongs to using her or his decryption key, and hence derive the secret key of the direct descendant roles. By recursively running this derivation process, the user can eventually recover the key of the role to which the message is encrypted. Hence she or he can decrypt the message.

When attaching a role to the role hierarchy, only the direct descendant roles of the role (if there are any) need to update their role public parameters, and none of the other roles and users are affected. This role needs to choose a new pseudo-identity and secret key because its old secret key might have been accessed by users from its previous ancestor roles. Since the hierarchy information is not used in the encryption algorithm, users of this role have access to the messages that were encrypted to its descendant roles before it became the ancestor role of these roles.

Adding users to roles does not affect the other roles and users either. The newly joined users have immediate access to the messages that were encrypted before they join the role, and the messages do not need to be re-encrypted. However, when revoking a user from the role, all the descendant roles will be affected; that is, all these roles will have to choose a new pseudo-identity and secret key, and update the role public parameters to prevent the revoked user from accessing any future messages encrypted to these roles.

We note that in the decryption algorithm of this construction, the number of times the *IBBE.Decrypt* algorithm is executed is linearly proportional to the distance between the role R and R' in the role hierarchical tree, which may be inefficient if the depth of the hierarchical tree is high. In addition, since each execution of the *IBBE.Decrypt* algorithm takes as an input the results from the previous execution, it is not possible to carry out these executions in parallel to improve the performance of the scheme.

3.4.2 Construction Type B

In this subsection, we give another construction of RBE described in Figure 3.2. In this construction, when encrypting a message to a set of roles $\mathcal{R}$, the message is not just directly encrypted to the roles in $\mathcal{R}$, it is also directly encrypted to all the ancestor roles of the roles in $\mathcal{R}$. The hierarchy relationships of the roles in $\mathcal{R}$ are reflected in the ciphertext. Therefore, a user does not need to run recursive algorithms to decrypt messages. Users in any role which have permissions to access messages can use the role secret key to decrypt messages, as messages are directly encrypted to the role.

Setup(λ): Generate system parameters $\{\mathsf{mk}, \mathsf{pk}\} \longleftarrow \mathcal{IBBE}.\mathcal{S}etup(\lambda)$.

Extract(mk, ID): When $\mathsf{ID} = \mathsf{ID}_U$ is an identity of a user U, the user decryption key is generated as $\mathsf{dk}_U \longleftarrow \mathcal{IBBE}.\mathcal{E}xtract(\mathsf{mk}, \mathsf{ID}_U)$. When $\mathsf{ID} = \mathcal{ID}_R$ is a set of pseudo-identities of a role R, the role secret keys are generated as $\mathsf{sk}_R^i \longleftarrow \mathcal{IBBE}.\mathcal{E}xtract(\mathsf{mk}, \mathsf{ID}_R^i)$ where $\mathsf{ID}_R^i \in \mathcal{ID}_R$.

ManageRole(mk, sk_R, ID_R, $\mathcal{T}$): To update the position of the role R with identity ID_R in the role hierarchy $\mathcal{T}$, the role public parameters are updated as $\mathsf{pub}_R^1 \longleftarrow \mathcal{PR}_R$. Here we let $\mathcal{PR}_R$ be the real identities of the role's ancestor roles.

Moreover, all the new descendant roles of the role R need to update their ancestor roles list to include the real identity of the role R; that is, for each descendant role R_k whose set of ancestor roles was $\mathcal{PR}_{R_k}$, the public parameters are updated as $\mathsf{pub}_{R_k}^1 \longleftarrow \mathcal{PR}_{R_k} \cup \mathsf{ID}_R$.

For the previous descendant roles (if there is any), their public parameters need to be updated to exclude the real identity of the role R.

AddUser(pk, sk_R, $\mathcal{U}_R$, ID_U): To add a user U with identity ID_U into a role R which has a set $\mathcal{U}_R$ of existing members, the public parameters pub_R^2 of the role R are updated as $\mathsf{pub}_R^2 \longleftarrow \mathcal{IBBE}.\mathcal{E}ncrypt(\mathsf{pk}, \mathcal{U}_R \cup U, \mathsf{sk}_R)$.

RevokeUser(pk, sk_R, $\mathcal{U}_R$, ID_U): To revoke a user U with identity ID_U from the role R which has a set $\mathcal{U}_R$ of existing members, first the role R chooses a new unused pseudo-identity ID_R and the corresponding secret key sk_R, and then updates the role public parameters $\mathsf{pub}_R^2 \longleftarrow \mathit{IBBE.Encrypt}(\mathsf{pk}, \mathcal{U}_R \backslash U, \mathsf{sk}_R)$. Other roles do not need to update their public parameters as the real identity of the role R does not change.

Encrypt(pk, $\mathcal{R}$, M): Assume there are n roles $\{R_1, \ldots, R_n\}$ in the set $\mathcal{R}$. Then we denote the ancestor role set $\mathcal{PR}_{\mathcal{R}} = \{\mathcal{PR}_{R_1}, \ldots, \mathcal{PR}_{R_n}\}$ where $\mathcal{PR}_{R_i}$ is the set of ancestor roles of the role R_i. When encrypting the message M to the set of roles $\mathcal{R}$, the ciphertext is output as $C \longleftarrow \mathit{IBBE.Encrypt}(\mathsf{pk}, \mathcal{R} \cup \mathcal{PR}_{\mathcal{R}}, M)$ where the pseudo-identities of the roles in the set are used in the encryption..

Decrypt(pk, R, dk_U, C): Assume that R is a role to which the message M is directly encrypted, and R' is a role that inherits permissions from R. Then a user U of the role R' decrypts the ciphertext C using the following algorithm.

```
set $\mathsf{sk}_{R'} \longleftarrow \mathit{IBBE.Decrypt}(\mathsf{pk}, \mathsf{dk}_U, \mathsf{pub}_{R'})$
output $M \longleftarrow \mathit{IBBE.Decrypt}(\mathsf{pk}, \mathsf{sk}_{R'}, C)$
```

Figure 3.2: Generic RBE Construction Type B

Putting a role into the role hierarchy requires all the descendant roles of the role to update their public parameters. The ancestor roles and all the users are not affected. Since the hierarchy relationships of roles are reflected in the ciphertext, users of the role cannot access messages that were encrypted to its descendant roles before the role is added.

Adding a new user to a role is the same as in the Construction Type A, and the new user can decrypt messages that were encrypted before she or he has joined the role. Since the hierarchy information is only used when encrypting messages, and the role public parameters store the real identities of ancestor roles, any change of an individual role will not affect other roles. Therefore, when revoking a user from a role, only the public parameters of this role need to be updated.

This generic construction dramatically simplifies the decryption process where the $\mathcal{IBBE}.\mathcal{D}ecrypt$ algorithm needs to be executed only twice. Moreover, revoking a user is efficient and does not affect any other roles. However, since the role public parameters are the identities of its ancestor roles, the size of the public parameters of each role is linearly proportional to the number of ancestor roles of the role.

3.4.3 Construction Type C

We now propose another RBE construction where a role can access messages that were encrypted to its new descendant role before the hierarchy changes and the number of times that the $\mathcal{IBBE}.\mathcal{D}ecrypt$ algorithm is executed is still independent of the role hierarchy. This construction is described in Figure 3.3.

Setup(λ): Generate system parameters $\{\mathsf{mk}, \mathsf{pk}\} \longleftarrow \mathcal{IBBE}.\mathcal{S}etup(\lambda)$.

Extract(mk, ID): When $\mathsf{ID} = \mathsf{ID}_U$ is an identity of a user U, the user decryption key is generated as $\mathsf{dk}_U \longleftarrow \mathcal{IBBE}.\mathcal{E}xtract(\mathsf{mk}, \mathsf{ID}_U)$. When $\mathsf{ID} = \mathcal{ID}_R$ is a set of pseudo-identities of a role R, the role secret keys are generated as $\mathsf{sk}_R^i \longleftarrow \mathcal{IBBE}.\mathcal{E}xtract(\mathsf{mk}, \mathsf{ID}_R^i)$ where $\mathsf{ID}_R^i \in \mathcal{ID}_R$.

ManageRole(mk, sk_R, ID_R, $\mathcal{T}$): To update the position of a role R with identity ID_R in the role hierarchy $\mathcal{T}$, first choose an unused pseudo-identity ID_R and the corresponding secret key sk_R, and then update the public parameters of the role R as $\mathsf{pub}_R^1 \longleftarrow \mathcal{IBBE}.\mathcal{E}xtract(\mathsf{pk}, \mathcal{PR}_R, \mathsf{sk}_R)$ and $\mathsf{pub}_R^2 \longleftarrow \mathcal{IBBE}.\mathcal{E}ncrypt(\mathsf{pk}, \mathcal{U}_R, \mathsf{sk}_R)$.

In addition, all the new descendant roles of the role R need to update their public parameters to have their permissions inherited by the new role. For each descendant role R_k whose set of ancestor roles was $\mathcal{PR}_{R_k}$, update the public parameters as $\mathsf{pub}_{R_k}^1 \longleftarrow \mathcal{IBBE}.\mathcal{E}xtract(\mathsf{pk}, \mathcal{PR}_{R_k} \cup \mathsf{ID}_R, \mathsf{sk}_{R_k})$.

Each previous descendant role R_i (if there is any) needs to choose a new unused pseudo-identity ID_{R_i} and the corresponding secret key sk_{R_i}, and then update the public parameters as $\mathsf{pub}^1_{R_i} \longleftarrow \mathcal{IBBE}.\mathcal{E}\mathit{xtract}(\mathsf{pk}, \mathcal{PR}_{R_i}, \mathsf{sk}_{R_i})$ where $\mathsf{ID}_R \notin \mathcal{PR}_{R_i}$, and $\mathsf{pub}^2_{R_i} \longleftarrow \mathcal{IBBE}.\mathcal{E}\mathit{ncrypt}(\mathsf{pk}, \mathcal{U}_{R_i}, \mathsf{sk}_{R_i})$.

AddUser(pk, sk_R, $\mathcal{U}_R$, ID_U): To add a user U with identity ID_U into a role R which has a set $\mathcal{U}_R$ of existing members, the public parameters pub^2_R of the role R are updated as $\mathsf{pub}^2_R \longleftarrow \mathcal{IBBE}.\mathcal{E}\mathit{ncrypt}(\mathsf{pk}, \mathcal{U}_R \cup U, \mathsf{sk}_R)$.

RevokeUser(pk, sk_R, $\mathcal{U}_R$, ID_U): In this construction, revoking a user affects all the descendant roles of the role. We denote a role set $\mathcal{R} = \mathcal{SR}_R \cup R$ where $\mathcal{SR}_R$ is the set of all the descendant roles of the role R, and the direct ancestor role set of each role $R_i \in \mathcal{R}$ as $\mathcal{PR}^D_{R_i}$.

When revoking a user U with identity ID_U from the role R which has a set $\mathcal{U}_R$ of existing members, first each role $R_i \in \mathcal{R}$ chooses a new unused pseudo-identity ID_{R_i} and the corresponding secret key sk_{R_i}, and then updates the role public parameters as $\mathsf{pub}^1_{R_i} \longleftarrow \mathcal{IBBE}.\mathcal{E}\mathit{ncrypt}(\mathsf{pk}, \mathcal{PR}^D_{R_i}, \mathsf{sk}_{R_i})$. Then R updates its public parameters as $\mathsf{pub}^2_R \longleftarrow \mathcal{IBBE}.\mathcal{E}\mathit{ncrypt}(\mathsf{pk}, \mathcal{U}_R \backslash U, \mathsf{sk}_R)$.

Encrypt(pk, $\mathcal{R}$, M): When encrypting a message M to a set of roles $\mathcal{R}$, the ciphertext is output as $C \longleftarrow \mathcal{IBBE}.\mathcal{E}\mathit{ncrypt}(\mathsf{pk}, \mathcal{R}, M)$.

Decrypt(pk, R, dk_U, C): Assume that R is a role to which the message M is directly encrypted, and R' is a role that inherits permissions from R. Then a user U of the role R' decrypts the ciphertext C using the following algorithm.

```
set $\mathsf{sk}_{R'} \longleftarrow \mathcal{IBBE}.\mathcal{D}\mathit{ecrypt}(\mathsf{pk}, \mathsf{dk}_U, \mathsf{pub}_{R'})$
set $\mathsf{sk}_R \longleftarrow \mathcal{IBBE}.\mathcal{D}\mathit{ecrypt}(\mathsf{pk}, \mathsf{sk}_{R'}, \mathsf{pub}_R)$
output $M \longleftarrow \mathcal{IBBE}.\mathcal{D}\mathit{ecrypt}(\mathsf{pk}, \mathsf{sk}_R, C)$
```

Figure 3.3: Generic RBE Construction Type C

In this construction, roles do not need to keep the identity list of their ancestor roles. Instead, they encrypt their role secret keys to ancestor roles. Therefore, when a role in the role hierarchy changes, the role manager of this role needs to choose a new pseudo-identity as its secret key might have been accessed by users from its previous ancestor roles. Since the role has access to the secret key of its descendant roles, users of this role have access to messages encrypted to its descendant roles before the role became the ancestor role of these roles.

The process of adding a user is the same as in the previous two constructions. Revoking a user requires all the descendant roles to choose new pseudo-identities and secret keys, and update the role public parameters to prevent the revoked user from accessing future messages encrypted to these roles.

Note that the decryption needs an additional $\mathcal{IBBE}.\mathcal{D}ecrypt$ execution in this construction, but the number of the executions is still independent of the depth of the role hierarchy.

3.5 Security Considerations

The intuition behind a KDM secure encryption scheme is that having access to encryptions of the secret keys does not help the adversary in breaking the security of the scheme. We note that this is exactly the case in our generic constructions where the public parameters are the encryptions of secret keys of IBBE schemes. If the adversary cannot learn anything from these role public parameters, then the security of a constructed RBE scheme is somehow equivalent to the security of the selected IBBE scheme. Therefore we have the following theorem.

Theorem 3.1 *Given a selective-ID ε_1-KDM secure IBBE scheme. An adversary's advantage $\mathsf{Adv}^{\mathcal{RBE}}$ in attacking a RBE scheme that is built based on our generic constructions using this IBBE scheme has $\mathsf{Adv}^{\mathcal{RBE}} < \varepsilon$.*

Proof: In our constructions, apart from the system public keys, the only information that can be publicly accessed is the encryptions of roles' secret keys and the ciphertexts.

Therefore, any attack to the RBE scheme can be disassembled into a KDM security game, where the function set $\mathcal{F}'$ that an adversary $\mathcal{A}'$ can use in *Queries* phase only contains functions in $SK = \{\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_p\}$ and $M = \{M_1, M_2, \ldots, M_q\}$, and SK is the set of all generated roles' secret keys and M is the set of all the messages that have been encrypted.

Now given an attacker $\mathcal{A}$ that wins the RBE security game with the advantage $\mathsf{Adv}^{\mathcal{RBE}}$, we construct another attacker $\mathcal{B}$ that can successfully win the KDM security game. $\mathcal{B}$ does the following:

Init: $\mathcal{B}$ outputs two sets of identities $\mathcal{U} = \{\mathsf{ID}_{U_1}, \cdots, \mathsf{ID}_{U_n}\}$ and $\mathcal{R} = \{\mathsf{ID}_{\mathcal{R}_1}, \cdots, \mathsf{ID}_{R_m}\}$ which $\mathcal{B}$ will query to the challenger and $\mathcal{A}$.

Setup: The challenger first executes $\mathcal{IBBE}.\mathcal{S}etup(\lambda)$ to setup the system and generate the master secret key mk and the public keys pk. Then the challenger sends the public keys pk to $\mathcal{B}$, and $\mathcal{B}$ forwards pk to the adversary $\mathcal{A}$. Then the challenger randomly chooses a bit $b \in \{0, 1\}$.

Queries: Assume that the adversary $\mathcal{A}$ chooses an identity $\mathsf{ID}_k \in \mathcal{R}$ as the role identity, and adaptively issues the following two types of queries to $\mathcal{B}$.

- *Extract*: $\mathcal{A}$ chooses a random identity $\mathsf{ID} \notin \mathcal{R}$ and ID has not been queried in the *Encryption* process, and sends the identity to the challenger. $\mathcal{B}$ sends the identity ID to the challenger, and the challenger computes and returns $\mathsf{sk}_{\mathsf{ID}} \longleftarrow \mathcal{IBBE}.\mathcal{E}xtract(\mathsf{mk}, \mathsf{ID})$ to $\mathcal{B}$. Then $\mathcal{B}$ sends the pair $(\mathsf{ID}, \mathsf{sk}_{\mathsf{ID}})$ to $\mathcal{A}$.
- *Encryption*: $\mathcal{A}$ chooses a set $\hat{\mathcal{U}} \subset \mathcal{U}$ of identities where none of the identities in $\hat{\mathcal{U}}$ either have been queried in the *Extract* processes or are in the set $\mathcal{R}$, and sends $\hat{\mathcal{U}}$ to $\mathcal{B}$. Then $\mathcal{B}$ passes the identity set $\hat{\mathcal{U}}$ and the identity ID_k to the challenger, and the challenger encrypts $\mathsf{sk}_{\mathsf{ID}_k}$ to the set $\hat{\mathcal{U}}$ and returns the ciphertext $c = \mathcal{IBBE}.\mathcal{E}ncrypt(\mathsf{pk}, \hat{\mathcal{U}}, \mathsf{sk}_{\mathsf{ID}_k})$ to $\mathcal{B}$, and $\mathcal{B}$ passes all the values which are returned by the challenger to the adversary $\mathcal{A}$.

Challenge: When the adversary $\mathcal{A}$ decides that all the queries have been completed, it chooses a function f which satisfies $f(\mathsf{sk}_1, \mathsf{sk}_2, \cdots, \mathsf{sk}_n) = M$ and sends the values to $\mathcal{B}$ and $\mathcal{B}$ forwards it to the challenger. The challenger chooses an identity $\mathsf{ID}_i \in \mathcal{U}$ and responds with $c = \mathsf{Enc}(\mathsf{pk}, \mathsf{ID}_i, M)$ if $b = 0$, or $c = \mathsf{Enc}(\mathsf{pk}, \mathsf{ID}_i, 0^{|M|})$ if $b = 1$.

Guess: Now assume that the adversary $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ by using the responses from the challenger in the above phases, and passes it to $\mathcal{B}$. If $\mathcal{A}$ has the right guess that $b = b'$, $\mathcal{B}$ wins the game.

In our RBE constructions, the *ManageRole* process has similar outputs as *AddUser*, and the only difference is the identity sets and role secret keys. So we omit the process for simplicity. From the above description, we see that if $\mathcal{A}$ wins the RBE security game, then $\mathcal{B}$ can win the KDM security game which implies $\mathsf{Adv}^{\mathcal{RBE}} = |Pr[b = b'] - \frac{1}{2}| < \epsilon$. Hence the theorem is proved.

In the next section, we use an example to illustrate and compare different types of generic constructions. For simplicity, in the following discussions of this section, we write $\mathcal{E}_{\mathcal{IBBE}}(M, \{\mathsf{ID}_1, \ldots, \mathsf{ID}_n\}) = \mathcal{IBBE}.\mathcal{E}ncrypt(\mathsf{pk}, \{\mathsf{ID}_1, \ldots, \mathsf{ID}_n\}, M)$.

3.6 Example Scenario and Comparison

Now consider the example role hierarchy shown in Figure 3.4. In Figure 3.4, the role R_1 inherits from role R_2 and role R_4, and the role R_2 inherits from R_5 and R_6. R_3 and R_4 both inherit from role R_7, and R_6 inherits from roles R_8 and R_9. Each role obtains a role secret sk_{R_i} from SA.

3.6.1 Role Public Parameters

First, we have a look at the role public parameters which are associated with each role. In our RBE constructions, each role has its public parameters $\mathsf{pub}_R = \{\mathsf{pub}_R^1, \mathsf{pub}_R^2\}$ where pub_R^1 is role hierarchy related parameters and pub_R^2 is user membership related parameters. Assume that R_i has a set of user members $\mathcal{U}_i$. The public parameters

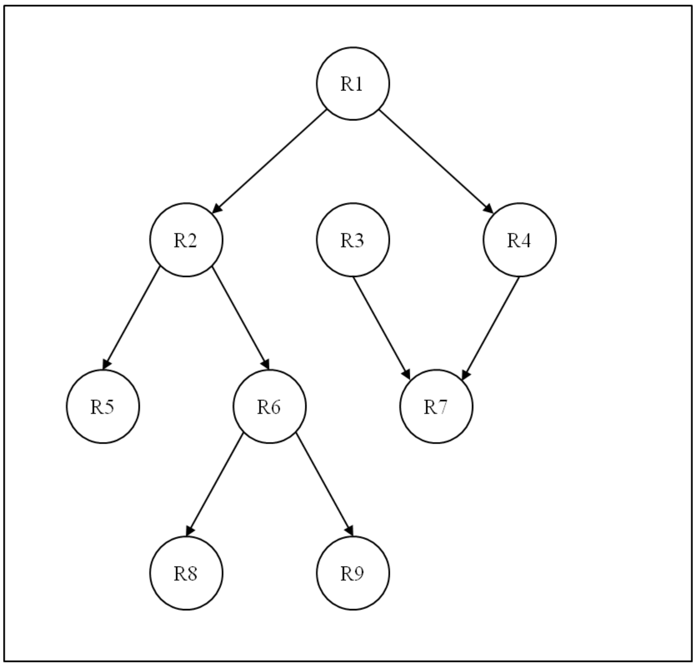


Figure 3.4: Hierarchical RBAC Example II

$\mathsf{pub}^2_{R_i}$ for this role will then be set as $\mathcal{IBBE.Encrypt}(\mathsf{pk}, \mathcal{U}_i, \mathsf{sk}_{R_i})$ which is the same in all the three constructions. $\mathsf{pub}^1_{R_i}$ for this role varies in different constructions. Table 3.1 shows the value of pub^1_R of each role for the example in different constructions.

In Table 3.1 we see that the role public parameters are constant in size, and are independent of the depth of the role hierarchy in Construction Types A and C. In Construction Type B, the size of role parameters of each role is linear to the number of its ancestor roles.

3.6.2 Encryption and Decryption

In this subsection, we describe the encryption and decryption algorithms of different constructions using the example shown in Figure 3.4. Assume that an owner encrypts a message M to role R_7 and R_9, and a user U with identity ID_U who is a member of role R_2 wishes to decrypt M. Since R_2 inherits from R_9, the user U is able to decrypt M.

Construction Type A. The owner runs $\mathcal{IBBE.Encrypt}(\mathsf{pk}, \{R_7, R_9\}, M)$ to encrypt

	Type A	**Type B**	**Type C**
R_1	$\varnothing$	$\{\mathsf{ID}_{R_1}\}$	$\varnothing$
R_2	$\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}_{R_2}, \{R_1\})$	$\{\mathsf{ID}_{R_1}, \mathsf{ID}_{R_2}\}$	$\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}_{R_2}, \{R_1\})$
R_3	$\varnothing$	$\{\mathsf{ID}_{R_3}\}$	$\varnothing$
R_4	$\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}_{R_4}, \{R_1\})$	$\{\mathsf{ID}_{R_1}, \mathsf{ID}_{R_4}\}$	$\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}_{R_4}, \{R_1\})$
R_5	$\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}_{R_5}, \{R_2\})$	$\{\mathsf{ID}_{R_1}, \mathsf{ID}_{R_2}, \mathsf{ID}_{R_5}\}$	$\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}_{R_5}, \{R_1, R_2\})$
R_6	$\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}_{R_6}, \{R_2\})$	$\{\mathsf{ID}_{R_1}, \mathsf{ID}_{R_2}, \mathsf{ID}_{R_6}\}$	$\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}_{R_6}, \{R_1, R_2\})$
R_7	$\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}_{R_7}, \{R_3, R_4\})$	$\{\mathsf{ID}_{R_1}, \mathsf{ID}_{R_3}, \mathsf{ID}_{R_4}, \mathsf{ID}_{R_7}\}$	$\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}_{R_7}, \{R_1, R_3, R_4\})$
R_8	$\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}_{R_8}, \{R_6\})$	$\{\mathsf{ID}_{R_1}, \mathsf{ID}_{R_2}, \mathsf{ID}_{R_6}, \mathsf{ID}_{R_8}\}$	$\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}_{R_8}, \{R_1, R_2, R_6\})$
R_9	$\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}_{R_9}, \{R_6\})$	$\{\mathsf{ID}_{R_1}, \mathsf{ID}_{R_2}, \mathsf{ID}_{R_6}, \mathsf{ID}_{R_9}\}$	$\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}_{R_9}, \{R_1, R_2, R_6\})$

TABLE 3.1: Role Public Parameters for Generic RBE Constructions

the message, and outputs the ciphertext as

$$C = \mathcal{E}_{\mathcal{IBBE}}(M, \{R_7, R_9\})$$

To decrypt the ciphertext C, the user U first decrypts the secret key sk_{R_2} of R_2 using the decryption key sk_U. Then she or he uses sk_{R_2} to decrypt sk_{R_6} of role R_6, and then decrypts sk_{R_9} of R_9 using sk_{R_6}, and lastly decrypts the message using sk_{R_9}.

Construction Type B. In this construction, the owner first retrieves the ancestor roles' identity list of role R_7 and R_9, and merges them into one list. Then the owner executes $\mathcal{IBBE}.\mathcal{E}ncrypt$ algorithm to output the ciphertext

$$C = \mathcal{E}_{\mathcal{IBBE}}(M, \{R_1, R_2, R_3, R_4, R_6, R_7, R_9\})$$

To decrypt the ciphertext C, the user U first decrypts the secret key sk_{R_2} of R_2 using the decryption key sk_U, and then decrypts the message directly using sk_{R_2}.

Construction Type C. The owner runs $\mathcal{IBBE}.\mathcal{E}ncrypt(\mathsf{pk}, \{R_7, R_9\}, M)$ to encrypt the message, and outputs the ciphertext as

$$C = \mathcal{E}_{\mathcal{IBBE}}(M, \{R_7, R_9\})$$

To decrypt the ciphertext C, the user U first decrypts the secret key sk_{R_2} of R_2 with the user decryption key sk_U, and then uses sk_{R_2} to decrypt sk_{R_9} of R_9. Then the user decrypts the message M using sk_{R_9}.

The decryption is less efficient in Construction Type A compared to the other two constructions, as a user needs to perform the IBBE decrypt algorithm every time she or he recovers the secret of the role one level down. In Constructions Types B and C, the user only performs IBBE decryption algorithm two and three times respectively, regardless of the depth of the role hierarchy.

3.6.3 Role Management

Let us now consider the role hierarchy management complexity. First, let us look at a scenario where a new role R_0 is added to the system, and it inherits from R_1. From Table 3.1, we can see that in Construction Type A, only the public parameters of role R_1 need to be changed to $\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}_{R_1}, \{R_0\})$, and all the other roles are not affected. In Construction Type B, all the roles need to update their public parameters to include role R_0 as one of their ancestor roles. However, the role public parameters in this construction are the identity list of all the ancestor roles, and they do not contain any secret values. Therefore the presence of a role manager is not compulsory in adding a new role. In Construction Type C, all the roles also need to update their public parameters to include role R_0 as their ancestor role. Since the role public parameters are generated using role secret keys, all these role managers have to be involved in generating the new role public parameters every time a new ancestor role of these roles is added to the system.

When a role is added to the role hierarchy, whether users of the new role are able to access messages encrypted to descendant roles before the new role has joined is another difference between the different construction types. Assume that a message M has been encrypted to R_1 before R_0 is added to the role hierarchy. With Construction Types A and C, messages are encrypted only to the target set of roles. Therefore, a change in role hierarchy does not affect the encrypted messages; that is, users of R_0 can access the message M once R_0 has been added as an ancestor role of R_1 without the need to re-encrypt the message M. However, in Construction Type B, the role inheritance relationships information has been used in the encryption of messages. Since R_0 was not an ancestor role of R_1 when the message M was encrypted to R_1 (and hence was not involved in the encryption of M), users of R_0 do not have access to the message M. If users of R_0 want to access the previously encrypted message M, this message needs to be re-encrypted.

Now let us consider a case where the role R_2 is removed from the system, and R_1 inherits directly from R_5 and R_6. In Construction Type A, all its descendant roles R_5, R_6, R_8 and R_9 need to choose new pseudo-identities and the corresponding secret keys. This ensures that R_2 does not have access to the role secret keys of these roles which will be used to decrypt future encrypted messages. R_5 and R_6 need to update their public parameters to $\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}'_{R_5}, \{R_1\})$ and $\mathcal{E}_{\mathcal{IBBE}}(\mathsf{sk}'_{R_6}, \{R_1\})$ respectively to allow the new direct ancestor role R_1 to access their new secret keys. Other descendant roles only re-encrypt their new role secret keys to the set of their ancestor roles. In Construction Type B, all its descendant roles only need to exclude the identity of R_2 from their public parameters, and it is clear that role managers do not need to be involved in this process. In Construction Type C, all the descendant roles need to choose new pseudo-identities and the corresponding secret keys for the same reason as in Construction Type A, and these roles also need to re-encrypt their new secret keys to their ancestor roles excluding R_2.

3.6.4 User Management

In this subsection, we discuss user management aspects. When a new user U has been granted membership of the role R_1, as we have described in section 3.4, the new user has immediate access to messages encrypted to R_1 and all descendant roles of R_1 without the need to re-encrypt these messages. This is the same in all the constructions. Only R_1 needs to update its public parameters, and no other roles and users are affected.

Assume that the role R_1 needs to revoke the user membership from an existing user U'. In Construction Types A and C, R_1 and all its descendant roles need to choose new pseudo-identities and the corresponding secret keys as the user U' might have accessed their role secret keys while she or he was a user of R_1, and hence these roles need to re-encrypt their new role secret. In Construction Type B, since users of roles do not

	Type A	Type B	Type C
- Size of role public parameters	$\mathcal{O}(1)$	$\mathcal{O}(p)$	$\mathcal{O}(1)$
- Decryption computation round	$\mathcal{O}(m)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
- Number of roles affected by adding a role	d	c	c
- Requiring use of the master secret when adding/removing a role	Yes	No	Yes
- Ability to decrypt messages that are encrypted before the new role is created	Yes	No	Yes
- Number of roles affected by a user revocation	$c + 1$	1	$c + 1$

Table 3.2: Comparison of Generic RBE Constructions

have access to role secret keys of other roles, U' only knows the role secret key of R_1. Therefore, only R_1 needs to choose a new pseudo-identity and update its role secret key and public parameters, and other roles are not affected.

3.6.5 Summary

We conclude by providing a comparison of results of our three generic RBE constructions in Table 3.2. In this table, a role has p ancestor roles, c descendant roles, d direct descendant roles and there are m roles on the path (including the ends) from this role to the role to which the messages are encrypted. From Table 3.2, it can be seen that each construction has different advantages and disadvantages.

Let us now consider several scenarios where these generic RBE constructions can be used. If a RBAC system requires efficient user revocation, the RBE construction type B has the most efficient user revocation and can be used. In the type B construction, when a new role is added to the system, even if it may inherit permissions from other roles, the users of this role cannot decrypt messages that were encrypted before the role was created.

Some RBAC systems may wish that users of a new role are able to decrypt any encrypted message to which they have the access no matter when the message was encrypted. In these cases, the RBAC systems can use the RBE constructions type A and type C. If such a system requires efficient decryption, the construction type C can be used as it has better decryption performance compared to type A where recursive computations are used.

In some RBAC systems, the privileges for role administration may be decentralised to different administrators. Each administrator manages a set of roles in the hierarchy which we call an administration domain. When a new role is added to the system, the administrator who manages the new role may not want other administrators to be affected by the operation. In such a case, the RBE construction type A can be used where only direct descendent roles will be affected by the operation.

From the above discussion, we can see that different constructions can be used for different purposes. In practice, the designer can select a suitable construction depending on the system requirements.

3.7 Conclusion

In this chapter, we have considered the formulation and the security properties of RBE schemes. We then proposed three different generic role-based encryption (RBE) constructions which can be used to enforce RBAC policies in an outsourcing environment. Compared to other existing cryptographic RBAC approaches, the RBE schemes constructed by the proposed generic solutions have common advantages, such as that a user only needs to keep constant size decryption keys regardless of the number of roles that the user has been assigned to. Depending on the chosen IBBE scheme, the constructed RBE scheme can have additional features, such as constant size ciphertext. We then compared these generic constructions, and analysed the advantages and disadvantages of each of these construction types. Different types of constructions can be used to build RBE schemes with different features, such as efficient user revocation, efficient decryption, and we have discussed the situations where different contributions can be used. These generic constructions would be helpful to the designer to choose a suitable RBE construction depending on the specific system requirements.

4

A Concrete Role-based Encryption Construction

In this chapter, we construct a concrete RBE scheme using a specific broadcast encryption mechanism described in [48]. In this scheme, the ciphertext and the decryption key that the user needs to keep is constant in size, and the user can be revoked from the role without affecting the owners and other users of the same role.

This chapter is organised as follows. Section 4.1 describes the Bilinear pairing which is used to construct the RBE scheme and the security problem that is used to prove the security of the RBE scheme. In section 4.2, we describe our RBE scheme. We give an analysis of the security and performance of our scheme in section 4.3. In section 4.4, we discuss some design aspects that can be optimised to achieve an efficient practical implementation of our scheme. Section 4.5 concludes the chapter.

4.1 Preliminaries

In this section, we first review the basic cryptographic principles that will be used throughout the chapter.

4.1.1 Bilinear Pairings

Let $\mathbb{G}_1$, $\mathbb{G}_2$ be two cyclic multiplicative 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 $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ 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 chapter, we require asymmetric bilinear groups, where the bilinear map takes inputs from two distinct isomorphic groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_1 \neq \mathbb{G}_2$. The reason that we choose asymmetric bilinear groups is that we keep the generator of one group as part of the master secret key. The benefit that we gain by defining two distinct groups is that our scheme can make use of certain families of non-supersingular elliptic curves defined in [103, 9].

4.1.2 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 ((P, Q, R, f)-GDDHE) by Boneh *et al.* [20]. The security of our scheme is based on this assumption. First, let us review the GDDHE problem.

Let p be an integer prime and s, n be positive integers. Let $P, Q, R \in \mathbb{F}_p[X_1, \ldots, X_n]^s$ be three s-tuples of n-variate polynomials over $\mathbb{F}_p$ and let $f \in \mathbb{F}_p[X_1, \ldots, X_n]$. We write $P = (p_1, p_2, \ldots, p_s)$, $Q = (q_1, q_2, \ldots, q_s)$ and $R = (r_1, r_2, \ldots, r_s)$, and the first components of P, Q, R satisfy $p_1 = q_1 = r_1 = 1$. For a set Ω, a function $h : \mathbb{F}_p \rightarrow \Omega$, and a vector $x_1, \ldots, x_n \in \mathbb{F}_p$, we write

$$h(P(x_1, \ldots, x_n)) = (h(p_1(x_1, \ldots, x_n)), \ldots, h(p_s(x_1, \ldots, x_n))) \in \Omega^s$$

Similar notation is used for the s-tuple Q and R. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be groups of order p and let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a non-degenerate bilinear map. Let $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ be generators of $\mathbb{G}_1, \mathbb{G}_2$ and set $g_t = e(g_1, g_2) \in G_T$.

A polynomial $f \in \mathbb{F}_p[X_1, \ldots, X_n]$ is said to be dependent on the sets (P, Q, R) if there exist $2s^2 + s$ constants $\{a_{i,j}\}_{i,j=1}^s$, $\{b_{i,j}\}_{i,j=1}^s$, $\{c_k\}_{k=1}^s$ $\in \mathbb{Z}_p$ such that

$$f = \sum_{i=1}^{s} \sum_{j=1}^{s} a_{i,j} p_i p_j + \sum_{i=1}^{s} \sum_{j=1}^{s} b_{i,j} q_i q_j + \sum_{k=1}^{s} c_k r_k$$

Definition 4.1 GDDHE Problem *(P, Q, R, f)-GDDHE Problem in $\mathbb{G}$ is defined as follows: Given a random $T \in \mathbb{G}_T$ and the vector*

$$H(x_1, \ldots, x_n) = (g_1^{P(x_1,\ldots,x_n)}, g_2^{Q(x_1,\ldots,x_n)}, g_t^{R(x_1,\ldots,x_n)}) \in \mathbb{G}_1^s \times \mathbb{G}_2^s \times \mathbb{G}_T^s$$

decide whether $T = g^{f(x_1,\ldots,x_n)}$.

The (P, Q, R, f)-GDDHE problem has been proved to have generic security when f is independent of (P, Q, R) by Boneh *et al.* [20]. In the subsequent security proof, we only need to give a specific P, Q, R and f and show that P, Q, R, f meet the requirement, as the (P, Q, R, f)-GDDHE problem has been proved to be hard for any choice of P, Q, R and f.

4.2 The RBE Scheme Construction

4.2.1 The RBE Scheme

In this section, we propose our specific RBE scheme. The scheme is designed as follows:

Setup(λ): Generate three groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of order p, and an bilinear map $\hat{e}$: $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Choose random generators $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$, a random value

$t \leftarrow \mathbb{Z}_p^*$, two secret values $s, k \leftarrow \mathbb{Z}_p^*$ and select two hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$, $H_2 : \mathbb{G}_T \rightarrow \mathbb{Z}_p^*$. The master secret key mk and system public key pk are defined as

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

where $w = g^s, v = \hat{e}(g, h), y = g^t$ and q is the maximum number of involved users and roles.

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

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

and gives dk_U to the user. 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 manager of R together with the user list $\mathcal{U}_R$ which is initially set to empty.

ManageRole(mk, sk_{R_i}, ID_{R_i}, $\mathcal{T}$): Let us consider the positioning of a role with identity ID_{R_i} in the role hierarchy. Assume that $\mathcal{PR}_{R_i} = \{\mathsf{ID}_{R_1}, \cdots, \mathsf{ID}_{R_m}\}$ is the set of the identities of all the ancestor roles of the role R_i, the role manager RM of the role R_i first outputs an empty role user list $\mathcal{RUL}_i$ and chooses a random secret value $r \leftarrow \mathbb{Z}_p^*$. Then RM computes

$$K_i = v^r, \quad W_i = w^{-r}$$

and sends K_i to SA via a secure channel. After receiving K_i, SA computes the role public parameters as

$$A_i = h^{(s+H_1(\mathsf{ID}_{R_i}))\prod_{j=1}^{m}(s+H_1(\mathsf{ID}_{R_j}))}, \quad B_i = {A_i}^k$$

$$S_i = \mathsf{sk}_{R_i} \cdot y^k \cdot y^{H_2(K_i)} = g^{\frac{1}{s+H_1(\mathsf{ID}_{R_i})} + t(k+H_2(K_i))}$$

AddUser(pk**,** pub_{R_i}**,** $\mathcal{N}$**,** ID_{U_k}**):** A user U_k with identity ID_{U_k} wishes to join the role R_i, and assume that R_i already has a set $\mathcal{N}$ of n users, where $U_k \notin \mathcal{N}$. RM first computes

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

Then RM adds ID_{U_k} into $\mathcal{RUL}_i$ and outputs the role public information

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

Encrypt(pk**,** pub_{R_x}**,** M**):** Assume that the owner of the message $M \in \mathbb{G}_T$ wants to encrypt M for the role R_x. Given $\mathsf{pk} = (w, v, y, g^k, h, h^s, \cdots, h^{s^m})$, the owner randomly picks $z \leftarrow \mathbb{Z}_p^*$ and computes and outputs the ciphertext $C = \langle C_1,\ C_2,\ C_3,\ C_4 \rangle$ as

$$C_1 = w^{-z}, \quad C_2 = y^{-z}, \quad C_3 = {A_x}^z, \quad C_4 = M \cdot v^{-z}$$

Decrypt(pk**,** pub_{R_i}**,** dk_{U_k}**,** C**):** Assume role R_x has a set $\mathcal{R}$ of ancestor roles, and the set $\mathcal{M} = R_x \cup \mathcal{R}$ has m roles $\{R_1, \cdots, R_m\}$. $R_i \in \mathcal{R}$ 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, and the user $U_k \in \mathcal{N}$ who is entitled to the role R_i wants to decrypt the message M. U_k computes

$$M' = C_4 \cdot (\hat{e}(C_1, h^{p_{i,\mathcal{M}}(s)}) \hat{e}(C_2, B_x) \hat{e}(\frac{S_i}{y^{H_2(K_i)}}, C_3))^{\frac{1}{\prod_{j=1, j\neq i}^{m} H_1(\mathsf{ID}_{R_j})}}$$

where

$$K_i = (\hat{e}(\mathsf{dk}_{U_k}, V_i) \cdot \hat{e}(W_i, h^{p_{k,\mathcal{N}}(s)}))^{\frac{1}{\prod_{j=1, j\neq k}^{n} H_1(\mathsf{ID}_{U_j})}}$$

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

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

We will show that $M = M'$ later.

RevokeUser(pk, pub_R, $\mathcal{N}$, ID_U)**:** To revoke a user U_k from a role R_i which has a set $\mathcal{U}_R$ of n users, and $U_k \in \mathcal{U}_R$, the role manager RM first removes ID_{U_k} from role user list $\mathcal{U}_R$ and chooses a random value $r'_i \leftarrow \mathbb{Z}_p^*$. Then RM computes

$$K'_i = v^{r'_i}, \quad W'_i = w^{-r'_i}, \quad V'_i = h^{r'_i \cdot \prod_{j=1, j\neq k}^{n}(s+H_1(\mathsf{ID}_{U_j}))}$$

and sends K'_i to SA via a secure channel. Suppose $\{R_1, \cdots, R_m\}$ are all the existing roles in the system, SA then chooses a random secret value $t' \leftarrow \mathbb{Z}_p^*$, and computes

$$y = g^{t'}, \quad S'_j = g^{\frac{1}{s+H_1(\mathsf{ID}_{R_j})}+t'(k+H_2(K_j))}, \;\; 1 \leq j \leq m$$

and publishes all these values. The role public information now changes to

$$(\mathsf{ID}_{R_i}, A_i, B_i, W'_i, V'_i, S'_i, \mathcal{RUL})$$

Correctness: To check the correctness of the scheme, now we show that $M = M'$. Assume that dk_{U_k} is a valid decryption key for identity ID_{U_k} of user U_k in the system, and U_k is the member of role ID_R which the message was encrypted to.

First, we look into the computation of K_i:

$$\begin{aligned} K_i^* &= \hat{e}(\mathsf{dk}_{U_k}, V_i) \cdot \hat{e}(W_i, h^{p_{k,\mathcal{N}}(s)}) \\ &= \hat{e}(g^{\frac{1}{s+H_1(\mathsf{ID}_{U_k})}}, h^{r_i \prod_{j=1}^{n}(s+H_1(\mathsf{ID}_{U_j}))}) \cdot \hat{e}(w^{-r_i}, h^{p_{k,\mathcal{N}}(s)}) \\ &= \hat{e}(g,h)^{r_i \prod_{j=1,j\neq k}^{n}(s+H_1(\mathsf{ID}_{U_j}))} \cdot \hat{e}(g,h)^{-r_i \cdot s \cdot p_{k,\mathcal{N}}(s)} \\ &= \hat{e}(g,h)^{r_i \cdot \prod_{j=1,j\neq k}^{n} H_1(\mathsf{ID}_{U_j})} \end{aligned}$$

Then we have

$$K_i = (K_i^*)^{\frac{1}{\prod_{j=1,j\neq k}^{n} H_1(\mathsf{ID}_{U_j})}} = \hat{e}(g,h)^{r_i} = v^{r_i}$$

Next we verify:

$$\begin{aligned}
K^* &= \hat{e}(C_1, h^{p_{i,\mathcal{M}}(s)}) \cdot \hat{e}(C_2, B_x) \cdot \hat{e}(S_i \cdot y^{-H_2(K_i)}, C_3) \\
&= \hat{e}(g^{-zs}, h^{p_{i,\mathcal{M}}(s)}) \cdot \hat{e}(g^{-zt}, h^{k\prod_{j=1}^{m}(s+H_1(\mathsf{ID}_{R_j}))}) \cdot \hat{e}(g^{\frac{1}{s+H_1(\mathsf{ID}_{R_i})}+kt}, h^{z\prod_{j=1}^{m}(s+H_1(\mathsf{ID}_{R_j}))}) \\
&= \hat{e}(g,h)^{-zs\cdot p_{i,\mathcal{M}}(s)+\frac{1}{s+H_1(\mathsf{ID}_{R_i})}\cdot z\prod_{j=1}^{m}(s+H_1(\mathsf{ID}_{R_j}))} \\
&= \hat{e}(g,h)^{z\cdot\prod_{j=1,j\neq i}^{m} H_1(\mathsf{ID}_{R_j})}
\end{aligned}$$

Thus, we have

$$M' = C_4 \cdot (K^*)^{\frac{1}{\prod_{j=1,j\neq i}^{m} H_1(\mathsf{ID}_{R_j})}} = Mv^{-z} \cdot \hat{e}(g,h)^z = M$$

In the above scheme, the owner of the message is able to specify a role R and encrypts the message M to this role. Only this role R and the roles which are ancestors of the role R can decrypt the message with their role secrets or randomised role secrets. For an individual role, the randomised role secret is encrypted in such a way that only the users in the role have the ability to decrypt the randomised role secret, and therefore are able to recover the message M. Clearly a user who cannot decrypt the randomised role secret cannot learn anything about the content of the message.

Note that the encryption has been split into separate levels. The role manager can add new users without changing the randomised role secret, which means that new users will be able to decrypt the message that was encrypted before she or he was assigned to this role, and the owner does not need to re-encrypt the message. On the other hand, when revoking a user, a role manager will need to have the SA re-randomise the role secret, so that this user cannot use the previous randomised role secret to decrypt messages any more, but no other roles will be affected by the changes to this role.

Let us now illustrate the scheme using some simple example scenarios.

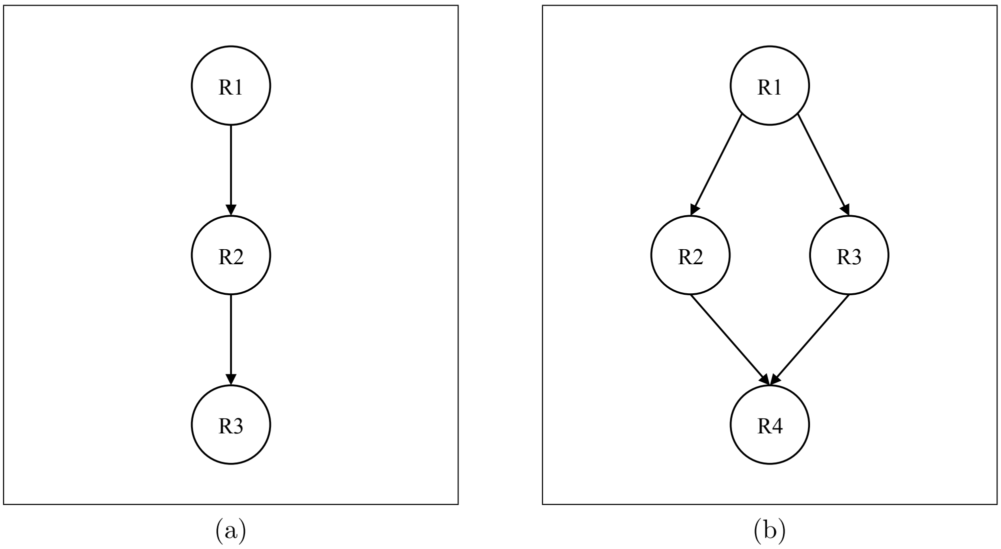


Figure 4.1: Hierarchical RBAC Example III

4.2.2 Example Scenarios

In this section, we give two small examples to illustrate the proposed RBE scheme (see Figure 4.1).

In Figure 4.1(a), the role R_1 is the ancestor role of role R_2, and the role R_2 is the ancestor role of role R_3. SA generates role public parameters pub_{R_i} for each role, where pub_{R_i} contain the identities of all the ancestor roles. When a user (owner of data) wishes to encrypt a message M for role R_3, pub_{R_3} will be used as the public key. Since pub_{R_3} includes the identities of R_1 and R_2, the message M is also encrypted for role R_1 and R_2.

When a user U_1 in role R_1 wants to decrypt the message, she or he only needs to use the user decryption key dk_{U_1} and the public parameters pub_{R_1} of role R_1 to decrypt the ciphertext. In this case, when the role manager wants to assign the role to a new user, RM only needs to update the pub_{R_1}.

Figure 4.1(b) illustrates a situation whereby the role R_1 is the ancestor role of two roles R_2 and R_3, and both R_2 and R_3 are ancestor roles of the role R_4. Firstly, SA generates role public parameters for each role. Note that pub_{R_4} will have the information for the identities of R_1, R_2 and R_3. Whenever a user encrypts a message M for role R_4, all the users in R_1, R_2 and R_3 will be able to decrypt the message. Here we can see that the size of pub_{R_4} remains constant regardless of the number of roles which inherit from it.

When adding a new role to the system, or removing an old role, only the descendent roles will be affected. In Figure 4.1(b), if SA wants to remove R_3 from the hierarchical structure, only the pub_{R_4} will need to be updated, and R_1 and R_2 will not be affected.

In a healthcare scenario, assume that Alice is a patient who wants to store her health record in an online personal health record (PHR) cloud server, so that she can easily share the information with other people she chooses. Now let us assume that Alice wishes to allow her doctors to access her PHR but not the nurses in the practice. In our hybrid RBE scheme, we first create the roles for Doctor and Nurse, and each role public key contains an encrypt form randomised role secret which can be used to decrypt the messages encrypted to the role itself. Alice simply encrypts her PHR to the role Doctor, and stores the ciphertexts in the online PHR cloud server. Assume that John is a doctor, and Jane is a nurse. Each of them will be given a decryption key.

Since Alice's PHR was encrypted to the role Doctor, John can use his decryption key with the public parameters of role Doctor to decrypt the randomised role secret of Doctor, and hence decrypt Alice's PHR. Because Alice's PHR was not encrypted for the role Nurse, Jane cannot use role Nurse's public parameters to decrypt the ciphertexts, so she will learn nothing about Alice's PHR. Let us assume that Tom is one of the technicians who is maintaining the service of the cloud. Though Tom can access the ciphertexts stored in the cloud, he cannot use any role public parameters to decrypt, as he is not assigned to any role.

Now assume that the hospital has created a new role Staff, which is a descendent role of role Doctor and Nurse. Let us assume that all the staff in the hospital will be added to this new role. The role Doctor and Nurse do not need to update their public parameters. If Alice has a message encrypted to the role Staff, John and Jane can both use their keys with the public parameters of role Doctor and Nurse to decrypt the Alice's message separately. When a new doctor joins the hospital, only the role Doctor needs to update the public parameters, and all the other roles will not be affected.

4.3 Analysis of Our RBE Scheme

In this section, we discuss our RBE scheme from the aspects of security and efficiency.

4.3.1 Security Analysis

We have defined the security properties of a RBE scheme in section 3.2. In this section, we analyse our scheme to show that it is chosen plaintext attack (CPA) secure.

Theorem 4.1 *The proposed RBE scheme is CPA secure against active adversary under the $GDDHE$ assumption in the Random Oracle Model.*

To prove Theorem 4.1, we start by defining a specific GDDHE problem based on the security assumption given in section 4.1.2, where we give a specific definition of P, Q, R and f which satisfies $f \notin (P, Q, R)$.

Definition 4.2 GDDHE Problem I *Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e})$ be a bilinear map group system and let g and h be the generator of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and $v = \hat{e}(g, h)$. Solving the $(P, Q, R, f) - GDDHE$ problem consists, given*

$$g, g^{\gamma}, \ldots, g^{\gamma^{2n}}, h, h^{\gamma}, \ldots, h^{\gamma^{2n}},$$

$$(g^{P(x_1,\ldots,x_n)}, h^{Q(x_1,\ldots,x_n)}, v^{R(x_1,\ldots,x_n)})$$

and $K \in \mathbb{G}_T$, in deciding whether K is equal to $\hat{e}(g, h)^{f(x_1,\ldots,x_n)}$ or some random element of $\mathbb{G}_T$.

Given an attacker $\mathcal{A}$ that wins the following game with probability $\mathsf{Adv}^{\mathcal{RBE}}_{IND-sID}$, we construct another attacker $\mathcal{B}$ that solves the $GDDHE$ problem.

Let q be the maximum number of identities of users and roles that the adversary can query, $\mathcal{U} = \{\mathsf{ID}_{U_1}, \ldots, \mathsf{ID}_{U_n}\}$ and $\mathcal{R} = \{\mathsf{ID}_{R_1}, \ldots, \mathsf{ID}_{R_m}\}$ are the set of users' and roles' identities respectively that the adversary will issue the queries. $\mathcal{B}$ will be given $g_0, g_0^s, \ldots, g_0^{s^{2n}}, h_0, h_0^s, \ldots, h_0^{s^{2n}}$, where s is a random number chosen by the challenger. We define the following polynomials,

- $g_1(x,k) = \prod_{i=1}^{k}(x + H_1(\mathsf{ID}_{R_i}))$
- $g_2(x,k) = \prod_{i=1}^{k}(x + H_1(\mathsf{ID}_{U_i}))$
- $f(x) = g_1(x,m) \cdot g_2(x,n)$
- $f_1(x,i) = \frac{f(x)}{x+H_1(\mathsf{ID}^*_{R_i})}$, where $i \in [1,m]$
- $f_2(x,i) = \frac{f(x)}{x+H_1(\mathsf{ID}^*_{U_i})}$, where $i \in [1,n]$

Init: The adversary $\mathcal{A}$ first outputs a set $\{\mathsf{ID}_1, \cdots, \mathsf{ID}_k\}$ of identities that he wants to attack (with $k \leq q$).

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

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

$$y = g^t = g_0^{t \cdot f(s)}, \quad \text{where } t \xleftarrow{R} \mathbb{Z}_p^*$$

Then $\mathcal{B}$ defines the public key as $\mathsf{pk} = (w, v, y, g^k, h, h^s, \cdots, h^{s^q})$, and creates the role parameters

$$A = h^{\prod_{i=1}^{m}(s+H_1(ID_{R_i}))} = h_0^{g_1(s,m)},$$

$$B = h^{k\prod_{i=1}^{m}(s+H_1(ID_{R_i}))} = h_0^{k \cdot g_1(s,m)}$$

Phase 1: The adversary $\mathcal{A}$ adaptively issues queries $q_1, \ldots, q_m$,

- If $\mathcal{A}$ issues a query on a user's identity ID_{U_j}, and ID_{U_j} has not been queried before, $\mathcal{B}$ computes the user decryption key as

$$\mathsf{sk}_{U_j} = g^{\frac{1}{s+H_1(\mathsf{ID}_{U_j})}} = g_0^{f_2(s,j)}$$

- If $\mathcal{A}$ has issued the query on ID_{U_j}, $\mathcal{B}$ then computes the role public parameters for role with ID_{R_i} as follows to assign the role to the user (t is the number of the users currently in the role),

$$K_i = v^{r_i}, \quad W_i = w^{-r_i} = g_0^{-r_i \cdot s \cdot f(s)}, \quad V_i = h^{r_i \cdot \prod_{j=1}^{t}(s+H_1(ID_{U_j}))} = h_0^{r_i \cdot g_1(s,t)}$$

$$S_i = y^{H_2(K_i)} \cdot g^{\frac{1}{s+H_1(ID_{R_i})}} \cdot g^{kt} = y^{H_2(K_i)} \cdot g^{f_1(s)+kt \cdot f(s)}$$

- If $\mathcal{A}$ issues a query on a role's identity ID_{R_i}, and ID_{R_i} has not been queried before, $\mathcal{B}$ computes the role secret and re-randomises as

$$\mathsf{sk}_{R_i} = g^{\frac{1}{s+H_1(\mathsf{ID}_{R_i})}+kt} = g_0^{f_1(s,i)+kt \cdot f(s)}$$

- If $\mathcal{A}$ has issued the query on ID_{R_i}, $\mathcal{B}$ then re-randomises the role secret as

$$\mathsf{sk}_{R_i} = g^{\frac{1}{s+H_1(\mathsf{ID}_{U_i})}+kt'} = g_0^{f_1(s,i)+kt' \cdot f(s)}$$

  and updates the system public parameter y to

$$y = g^{t'} = g_0^{t' \cdot f(s)}$$

As we use the ID-based broadcast encryption to encrypt the re-randomised role secret, and this IBBE scheme has been proven to be secure in [48], we assume this to be the case here as well, so the adversary will learn nothing about S_i here.

Challenge: Once $\mathcal{A}$ decides that Phase 1 is over, it publishes the identity ID_{R_i} of the role to which it wishes to encrypt the message and two messages M_0, M_1 on which it wishes to be challenged. Then $\mathcal{B}$ simulates *Encrypt* algorithm by constructing the ciphertext for the message M_b for a random $b \in \{0, 1\}$ as,

$$C_1 = w^{-z} = g_0^{-z \cdot s \cdot f(s)}, \ , C_2 = y^{-z} = g_0^{-z \cdot t \cdot f(s)}$$

$$C_3 = A^z = h_0^{z \cdot g_1(s,m)}, \ C_4 = M_b \cdot \hat{e}(g_0, h_0)^{-z \cdot f(s)}$$

Phase 2: The adversary $\mathcal{A}$ adaptively issues more queries $q_{m+1}, \ldots, q_n$ which does the same as the steps in **Phase 1**.

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

Then we have

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

Chosen-Ciphertext Security: Our security definition and proof are in the chosen-plaintext model. However, our scheme can be extended to the chosen-ciphertext model by using a standard transformation in [57]. Let $\mathcal{E}$ be a probabilistic public key encryption scheme, and $\mathcal{E}_{pk}(M; r)$ be the encryption of M using the random bits r under the public key pk. A hybrid scheme $\mathcal{E}^{hy}$ is defined in [57] as:

$$\mathcal{E}^{hy}_{pk}(M) \;=\; \mathcal{E}_{pk}(\delta; H(\delta, M)) \parallel G(\delta) \oplus M$$

where H and G are two hash functions $H : \mathsf{MSPC} \times \{0,1\}^n \to \{0,1\}^n, G : \mathsf{MSPC} \to \{0,1\}^n$ and δ is randomly chosen from MSPC. In [57], the authors show that if $\mathcal{E}$ is a one-way encryption scheme then $\mathcal{E}^{hy}$ is a chosen ciphertext secure system (IND-CCA) in the random oracle model. Since semantic security implies one-way encryption, this result also applies if $\mathcal{E}$ is semantically secure (IND-sID-CPA).

Now we apply this generic conversion to our proposed CCA secure RBE scheme and the chosen-ciphertext secure RBE scheme is as follows:

Setup(λ): Same as in the original RBE scheme with two additional hash functions $H_3 : \mathbb{G}_T \times \{0,1\}^n \;\to\; \mathbb{Z}_p^*,\; H_4 : \mathbb{G}_T \;\to\; \{0,1\}^n$.

Extract(mk, ID)**:** Same as in the original RBE scheme.

ManageRole(mk, sk_R, ID_R, $\mathcal{T}$)**:** Same as in the original RBE scheme.

AddUser(pk, sk_{R_i}, $\mathcal{N}$, ID_{U_k})**:** Same as in the original RBE scheme.

Encrypt(pk, pub_R, M)**:** Assume that the owner of the message $M \in \{0,1\}^n$ wants to encrypt M for the role R_x. Given $\mathsf{pk} = (w, v, y, g^k, h, h^s, \cdots, h^{s^m})$, the owner randomly picks $\delta \leftarrow \mathbb{G}_T$, sets $z = H_3(\delta, M)$ and computes

$$C_1 = w^{-z}, \quad C_2 = y^{-z}, \quad C_3 = {A_x}^z, \quad C_4 = \delta \cdot v^{-z}$$

The owner outputs the ciphertext

$$C = \langle C_1,\ C_2,\ C_3,\ C_4,\ H_4(\delta) \oplus M \rangle$$

Decrypt(pk, pub_R, dk_{U_k}, C)**:** Let $C = \langle C_1, C_2, C_3, C_4, C_5 \rangle$ be a ciphertext. The user first computes δ as in the original scheme, and then computes

$$M = C_5 \oplus H_4(\delta)$$

RevokeUser(pk, sk_R, $\mathcal{N}$, ID_U)**:** Same as in the original RBE scheme.

4.3.2 Scheme Efficiency

Our scheme has several distinct advantages compared with the schemes that are earlier mentioned in the Related Work in section 2.3.

Size of Ciphertext and Decryption Key

Our scheme has constant-size ciphertext and decryption keys. That is, the size of the ciphertext and decryption key is independent of the number of roles and users in the system, and it will not increase when new roles are created or more users join the system. These are significant features when it comes to development of large-scale systems. In comparison, for the schemes given in [126, 12], the size of the ciphertext is linearly proportional to the number of users, which make these schemes inefficient in practical systems where there can be a large number of users. Furthermore, in these schemes, when a user has multiple roles, multiple copies of the keys or ciphertexts are required for a single user in every role, while in our scheme, no matter how many roles a user has, the size of keys and ciphertexts remain constant. The scheme in [148] has reduced the size of the ciphertext to be linearly proportional to the depth of the role hierarchy. However their scheme is not able to deal with user revocation, and the ciphertext size is still not constant.

User Management

The schemes mentioned in the Related Work in section 2.3 all assume the existence of a single trusted party who manages the role memberships of the users. In this case, this single trusted party needs to verify the qualifications of the user whenever a role wishes to add a new user. However, in the real world, whether a user is entitled to a role or not is usually decided by different entities who are responsible for validating the users' qualifications for different roles. In our scheme, the role memberships of the users are no longer decided by a single trusted party. Instead, they are controlled by role managers, who could be different for each role. This is more natural and reflects the situation in practice. A role manager can add or revoke a user without having to gain admissions from other parties, including the trusted party.

As mentioned earlier, in our scheme, a role manager RM can exclude the user from accessing future encrypted data. In [148], once the user obtains the decryption key, the manager cannot revoke the user's permission even if the user does not qualify for the role, as the decryption relies on the hierarchical structure of the roles. Unless the ancestor roles have changed, the user can always decrypt the messages. In addition, when a user has been excluded in our scheme, the other users of the same role will not be affected. In the scheme described in [126], as the key structure had been constructed based on the access matrix, when a user is removed from the access matrix, the key management structure needs to be updated as well. This in turn will change the keys of all the other users. In the scheme [12], as the ciphertext is associated with the

	Our scheme	[126]	[12]	[148]
Ciphertext length	$\mathcal{O}(1)$	$\mathcal{O}(\mathrm{n})$	$\mathcal{O}(\mathrm{n})$	$\mathcal{O}(\mathrm{m})$
Encryption complexity	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\mathrm{n})$	$\mathcal{O}(\mathrm{n})$
Decryption complexity	$\mathcal{O}(\mathrm{n})$	$\mathcal{O}(1)$	$\mathcal{O}(\mathrm{n})$	$\mathcal{O}(\mathrm{n})$
Computation Round(s) in Decryption	1	m	m	1
Users revocation	Yes	Yes	Yes	No
Users affected in a revocation	0	n	n	

Table 4.1: Comparison of Cryptographic RBAC Schemes

hierarchical structure, the system parameters need to be re-generated every time a user's permission is revoked, which results in all the other users having to update their keys. In our scheme, when a user is excluded, we can see that only roles need to update their public information.

Computations

Table 4.1 shows a comparison of our scheme with the other schemes in [126, 12, 148]. The Table assumes that there are n users of the same role and there are m roles between the role of the user and the specified role for the encryption in the role hierarchical structure.

The decryption algorithm in our scheme only requires one round of computation, while in [126, 12], the user needs to compute the secrets of all the roles (nodes) which are on the path from the entitled role to the target one in the policy tree.

As shown in Table 4.1, though the scheme in [126] is computationally more efficient in terms of decryption, the complexity of role creation, user grant and revocation operations is linearly proportional to the number of authorised users. This makes the scheme not scalable in practice. Moreover, the number of computation rounds in decryption is linearly proportional to the number of roles in between in the role hierarchical structure. However our scheme, with its constant size ciphertext and decryption key, is more efficient for large-scale systems, and the decryption only requires one computation round. Comparing our scheme to the other two schemes in [12, 148], our scheme has similar performance but has constant size ciphertext and decryption keys. Furthermore, our scheme also supports dynamic user revocation while the other two schemes do not. Effective dynamic user revocation is a fundamental requirement in most practical systems.

4.4 Discussion

In this section, we briefly consider some aspects that can be optimised in the implementation of the proposed RBE scheme.

Preventing malicious cloud provider

When an owner of the data wants to encrypt private data, she or he needs to obtain the system public parameters $w, v \in \mathsf{pk}$ and role public information $A_R \in \mathsf{pub}_R$ from the cloud in order to run the *Encrypt* algorithm. This process can be susceptible to a man-in-the-middle attack because a malicious cloud provider can return the pub'_R of another role R' instead of the real pub_R of the role R that the owner wants to encrypt the private data with. If this is done, then the malicious cloud provider will be able to reveal the content of data without the knowledge of the owner.

In order to prevent this kind of attack, we propose the following: first we make the SA return system parameters w, v together with the decryption key to the user while executing the *Extract* algorithm. Then we let SA sign the value A_R under the identity of role R using an appropriate ID-based signature scheme (eg. [71, 35]), and return the signature $\mathsf{sig}(A_R, \mathsf{ID}_R)$ along with A_R. Upon receiving A_R in running the *Encrypt* algorithm, the owner should verify the signature corresponding to the identity of the role, to which she or he wants to encrypt the private data, before sending the ciphertext to the cloud. If the verification fails, it means that the value A_R is not the public information of the specified role R and the owner should not use it to encrypt the data.

Decrypting previously generated messages

After executing the *RevokeUser* algorithm in our RBE scheme, the role public information pub_R is updated to prevent the excluded user from decrypting any future messages.

However, the *Decrypt* algorithm uses pub_R which means the messages encrypted before executing *RevokeUser* algorithm cannot be decrypted even by the existing users of the role because the pub_R used to decrypt the previously generated messages has been replaced.

We can allow for this and enable decryption of previously generated messages in our RBE scheme by simply creating the indices for the ciphertext C and role public information pub_R. When RM runs the *AddUser* or *RevokeUser* algorithm to create or update pub_R, the algorithm will create a unique index for pub_R, and insert it in the cloud rather than replacing the previous pub_R. When an owner of the data is encrypting the private data, the *Encrypt* algorithm can be modified to attach the index of the latest version pub_R to the ciphertext C.

When a user wants to decrypt a ciphertext C, she or he can ask the cloud provider to return the ciphertext C and pub_R of the role. The cloud provider can easily choose and return the corresponding pub_R according to the index included in the ciphertext C.

Moreover, the role public information's update in *RevokeUser* algorithm requires the changes in the public parameter S of all the roles in the system, and it could be inefficient when there are a large number of roles. However, this work can be delegated to a trust authority by SA without giving out the master secret. Such an approach is being addressed as part of the implementation of the scheme.

Optimising decryption strategy

In our RBE scheme, let us assume that m is the number of ancestor roles of a specific role, and assume that there are n users in the same role with the user who runs the *Decrypt* algorithm. The *Decrypt* algorithm requires the expansion of two polynomials of degrees $m + 1$ and $n + 1$ respectively. This calculation could be time consuming if m and n are very large numbers.

We note that these two polynomials remain the same in the decryption of two different messages if the identities of roles and users are not changed. Therefore, in implementation, the user can keep these values as auxiliary information to help with decrypting messages. These values only need to be re-calculated when the ancestor roles of the specific role are changed or the permission is revoked from another user in the same role.

In addition, the user needs the system public keys $\mathsf{pk} = (w, v, g^k, h, h^s, \cdots, h^{s^q})$ to calculate the expansion of the polynomials, which is inefficient in practice because downloading the pk every time could cost network traffic if q is a very large number. We can then utilise the computing power of the cloud to do the polynomial expansion because calculating them does not require any secret values, and the cloud only needs to return the result of the polynomial expansion to the user, which can dramatically simplify the work for the user as well.

More precisely, in the decryption algorithm, the user in role R_1 wants to decrypt the message which is encrypted to role R_2, and R_1 inherits all the permissions of R_2. When the user retrieves the ciphertext from the cloud, the cloud computes the value of $h^{p_{i,\mathcal{M}}(s)}$ and $h^{p_{k,\mathcal{N}}(s)}$ for the user and returns them to the user. This user can keep two auxiliary values (Aux_R, Aux_U), where

$$Aux_R = h^{p_{i,\mathcal{M}}(s)}, \quad Aux_U = H_2(K_i)$$

If a user wants to decrypt other messages which are encrypted to role R_2 in future, Aux_R can be reused in *Decrypt* algorithm, and the user only needs to ask the cloud to re-calculate the value when one or more ancestor roles of R_2 are changed. Similarly, the user can reuse Aux_U to decrypt any future messages, and Aux_U only needs to be re-calculated when one or more users are excluded from R_1. This "caching" strategy can improve the decryption performance substantially as the user only needs to compute three pairings most of the time.

Implementation Issues

After the users have been given the decryption keys (when they join the system), the users should take the responsibility to store their secret keys safely. If a user's secret key is revealed to others, other parties will have the ability to decrypt any data on behalf of the actual user. If a user discovers that her or his secret key has been revealed to other users, she or he should report this to all appropriate role managers immediately and request that this user's identity be excluded from the role.

Before a user is included into a role, the role manager will need to authenticate the user so that the role manager can be convinced that the user qualifies for the role. We have not considered the authentication mechanisms in this chapter; we have assumed that such mechanisms exist and assume that the role manager will assign the role only to qualified users in the system.

In our scheme, the cloud platform is only used for storage purposes. The encryption and decryption computations do not occur in the cloud; hence the original private data from the owner and the secret key of a user will not be given to the cloud. Even if the cloud may be involved in the decryption algorithm, the computations in the cloud will not involve any secret values from either the owners or the users.

In our RBE scheme, the system public key pk is stored publicly in the cloud, and can be retrieved by any user, who has access to the cloud, to encrypt the data. However in the case where a user needs to identify the original source of the data, we can simply employ an ID-based digital signature scheme (e.g. [71, 35]), and have the owner sign the ciphertext. Then the ciphertext can be stored along with its signature. When a user wants to decrypt the ciphertext, she or he will need to verify the signature to check if the ciphertext is correctly signed by the owner from whom the user is expecting the data.

Comparison with Constructions in Chapter 3

In Chapter 3, we have described three generic RBE constructions and provided a comparison of them. In this chapter, we used a different approach to build the RBE scheme in order to show that a specific RBE scheme can have features that are different from the ones in the generic constructions. To compare the differences of them, we reuse the result from the Table 3.2 and include the concrete RBE scheme in the comparison in Table 4.2. In this table, the construction "RBE" refers to the concrete RBE scheme. Same as in the Table 3.2, we assume that a role has p ancestor roles, c descendant

	Type A	Type B	Type C	RBE
• Size of role public parameters	$\mathcal{O}(1)$	$\mathcal{O}(p)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
• Decryption computation round	$\mathcal{O}(m)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
• Number of roles affected by adding a role	d	c	c	c
• Requiring use of the master secret when adding/removing a role	Yes	No	Yes	Yes
• Ability to decrypt messages that are encrypted before the new role is created	Yes	No	Yes	No
• Number of roles affected by a user revocation	$c+1$	1	$c+1$	n
• Constant size ciphertext				Yes
• Constant size decryption key				Yes

Table 4.2: Comparison of RBE Constructions

roles, d direct descendant roles, and there are m roles on the path (including the ends) from this role to the role to which the messages is encrypted, and n roles in total in the hierarchy. From the Table 4.2, we can see that the RBE scheme described in this chapter has different features compared to the generic RBE constructions. In addition, the concrete RBE scheme shows that constant size ciphertext and decryption keys can be achieved by choosing a proper IBBE scheme. Given the differences of these RBE constructions, a RBAC system can choose a suitable RBE scheme depending on the requirements of the system.

4.5 Conclusion

In this chapter, we have proposed a Role-based Encryption (RBE) scheme that combines RBAC with encryption to address security requirements for storage of information in the cloud. We have constructed a specific RBE scheme using the broadcast encryption scheme described in [48]. We have conducted security analysis of our scheme and have given proofs to show that our scheme is secure against adaptive attack. We have discussed the performance and efficiency of our scheme and have compared it with other previously related work. We have shown that our scheme has several superior characteristics such as constant size ciphertext and decryption keys, efficient user revocation and user management, and the ability to handle role hierarchies. We have also considered some aspects that can be optimised to achieve efficient implementation. We believe that the proposed scheme is suitable for large-scale systems, especially in the context of achieving user centric secure information storage in a cloud computing environment.

5

A Secure Cloud Storage System based on Role-based Encryption

In this chapter, we present the design of a secure RBAC-based cloud storage system where the access control policies are enforced by an improved RBE scheme with an efficient user revocation based on the RBE scheme described in Chapter 4. With the improved 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.

We have developed a secure cloud storage system using an improved RBE scheme and hybrid cloud architecture. The most frequently used system operations such as encryption of data by a data owner and 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 shown 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.

This chapter is organised as follows. Section 5.1 briefly describes the structure of our hybrid system. In Section 5.2, we introduce the improved RBE scheme and an ID-based signature scheme that will be used in the development of the secure cloud storage system. The architecture for our secure cloud storage system is presented in section 5.3. Section 5.4 describes the methods that are used in our implementation, and gives the result of our experiments. Section 5.5 concludes the chapter.

5.1 System Overview

Data privacy has become a critical issue when people consider using the cloud to store their private or sensitive data. Several recent surveys [78, 87] show that 88% of potential cloud consumers are worried about the privacy of their data, and security is often cited as the top obstacle for cloud adoption. Since a private cloud is usually operated by a single organisation, it is considered to be more secure than a public cloud. A recent survey [7] 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. However, storing all data in a private cloud will sacrifice the benefits that have been brought by the public cloud.

In this chapter, we develop a secure cloud data storage architecture using a hybrid

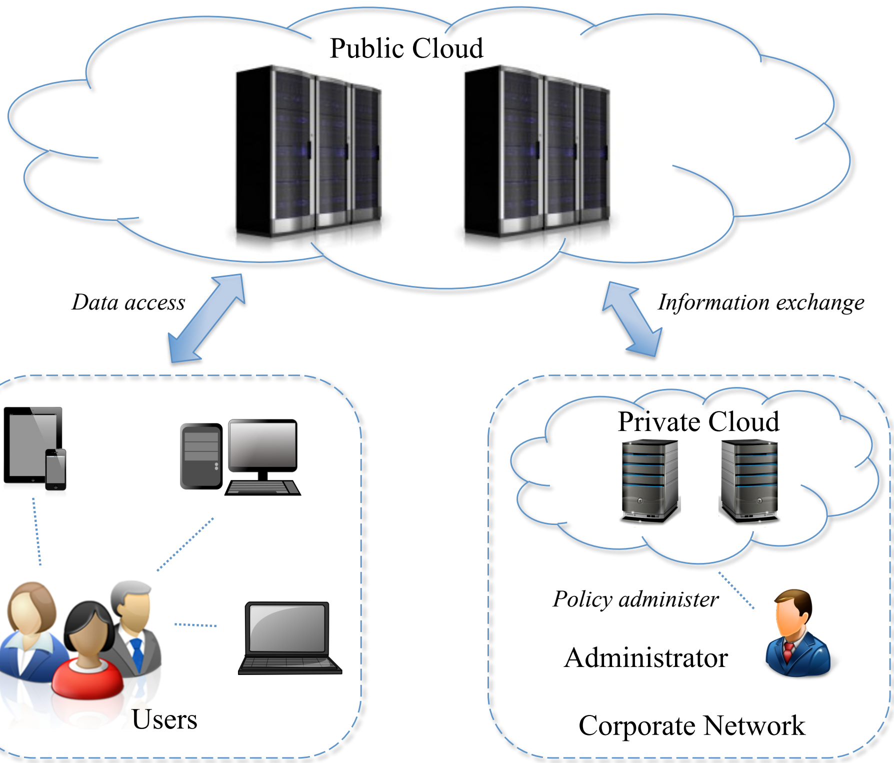


Figure 5.1: Hybrid Storage Cloud Overview

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 Figure 5.1. 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 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 the 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.

5.2 Cryptographic Schemes

In this section, we describe the cryptographic schemes which we use in building the secure data storage cloud. We first propose an improved RBE scheme based on the one we described in Chapter 4 to protect the stored data in the system. Then we describe a variant ID-based signature scheme, which is derived from the scheme introduced in [10], for the authentication purpose in the system.

5.2.1 An Improved Role-based Encryption Scheme

In order to develop our secure cloud storage system, we propose an improved RBE scheme which is designed using asymmetric bilinear groups. In this section, we present this improved RBE scheme which is used in our cloud storage system to enforce the RBAC policies.

We use an asymmetric bilinear pairing which takes inputs from two distinct isomorphic groups in this scheme, so that a wider range of curves is able to be used in our system. Assume that these two input groups are denoted as $\mathbb{G}_1$, $\mathbb{G}_2$ and 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 chapter, we will make use of this characteristic to improve the performance of our RBE scheme.

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 manager of R together with the $\mathcal{RUL}_R$ which is initially set to empty.

ManageRole(mk, ID_R, $\mathcal{T}$)**:** 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}, M)**:** 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

$$C_1 = w^{-z}, \quad C_2 = {A_x}^z, \quad C_3 = {B_x}^z, \quad K = v^z$$

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

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

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

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.

As we discussed previously, the **Decrypt** algorithm requires the expansion of two polynomials of m (the number of ancestor roles) and n (the 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 inputs 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.

Compared to the RBE scheme described in Chapter 4, this RBE scheme has made a few improvements. First, this RBE scheme switches the use of $\mathbb{G}_1$ and $\mathbb{G}_2$ which results in a better performance in implementations. Second, the user revocation in this scheme requires only one role to update its public parameters in contrast to all the roles in the RBE scheme in Chapter 4. In addition, in this scheme, role managers of individual roles can revoke users without the need of having the administraor involved.

Security Analysis

Chosen Plaintext Security. Similarly, we first prove CPA security of our RBE scheme based on the GDDHE assumption.

Theorem 5.1 *The proposed RBE scheme is chosen plaintext secure under the $GDDHE$ assumption.*

To prove Theorem 5.1, we start by defining another specific GDDHE problem where it clearly shows that $f = czh(s)$ is independent from P and Q. In the following definition, s, k, c, z, t are random elements chosen from $\mathbb{Z}_p^*$ as described in our RBE scheme.

Definition 5.1 GDDHE Problem II *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*

$$g_0, g_0^s, \ldots, g_0^{s^{m-1}},\ g_0^{c \cdot g(s)},\ g_0^{ckg(s)},\ g_0^{ctg(s)},\ g_0^{cktg(s)},\ g_0^{ch(s)}$$

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

in deciding whether K is equal to $\hat{e}(g_0, h_0)^{czh(s)}$ 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 5.1. 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

$$g = g_0^{c \cdot g(s)},\ w = h_0^{s \cdot f(s)},\ w^s = h_0^{s^2 \cdot f(s)},\ v = \hat{e}(g, h) = \hat{e}(g_0, h_0)^{c \cdot h(s)}$$

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 $q_1, \ldots, q_k$

- *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
$$W = h_0^{-r'sf(s)} = w^{-r'},$$
$$V = g_0^{cr'h(s)} = g^{r'f(s)} = g^{r' \cdot \prod_{i=1}^{n}(s+H_1(\mathsf{ID}^*_{U_i}))}$$
$$S = H_2(\hat{e}(g_0, h_0)^{r'c \cdot h(s)}) \cdot g_0^{g_i(s)} \cdot g_0^{kt'c \cdot g(s)}$$
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.

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

$$S = H_2(\hat{e}(g_0, h_0)^{rc \cdot h(s)}) \cdot g_0^{g_i(s)} \cdot g_0^{ktc \cdot g(s)}$$

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

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

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

Note that if $K' = \hat{e}(g_0, h_0)^{zc \cdot h(s)}$, 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 the above, we have proved that the proposed RBE scheme is chosen plaintext secure. Generic conversion methods [33, 24] exist to convert a CPA-secure scheme to a CCA-secure scheme. Our RBE scheme can be extended to a CCA-secure scheme using a similar approach, such as using strongly unforgeable signatures.

5.2.2 An Extended Scheme Supporting Multiple-role Encryption

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 improved RBE scheme which supports the encryption for multiple roles. In the extended scheme, only the **Encrypt** and **Decrypt** algorithms are modified from the ones in the improved scheme, and the other algorithms are the same as in the improved 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 improved RBE scheme, the extended RBE scheme inherits most of the features from the improved 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 at a similar level to the size of plaintext which we will show in section 5.4.

5.2.3 An ID-based Signature Scheme

In this section, we describe the IBS scheme proposed in [10]. However, in order to make the scheme cater for the same key format as the RBE scheme in our system, we introduce 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 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 $h \in \mathbb{G}_2$, a secret value $s \leftarrow \mathbb{Z}_p^*$ and two hash functions $H_1 : \{0,1\}^* \to \mathbb{Z}_p^*$, $H_2 : \{0,1\}^* \times \mathbb{G}_T \to \mathbb{Z}_p^*$. 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

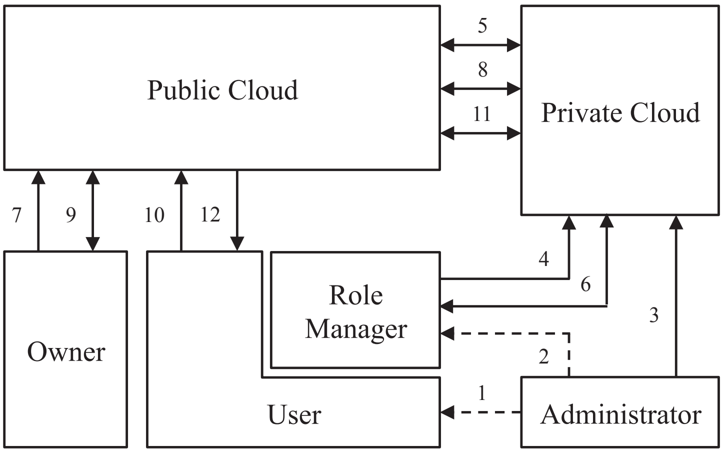


Figure 5.2: Hybrid RBE System Architecture

holds

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

5.3 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 are 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.

5.3.1 Architectural Components

We first consider the components of the system architecture shown in Figure 5.2. The numbers shown in the figure refer to the system operations which will be described in section 5.3.2.

Public Cloud : Public cloud is a third party cloud provider which resides outside the infrastructure of 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 she or he communicates with the public cloud. In this case, the 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 thesis; hence in this chapter, 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 : A 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 compared to the data stored in the 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 as well as the public cloud, and users do not have direct access to it. 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 decision making.

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. 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 the 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.

Administrator: The administrator is the system administrator 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 role hierarchy, 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 an 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[123], and he gave the first construction of IBS. Since then, many different IBS schemes have been proposed. We have chosen the scheme proposed in [10] in our system implementation, as it has a similar key format to our RBE scheme. We use the IBS scheme described in section 5.2.3 in our implementation as it has the same key format as our proposed RBE 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.

5.3.2 System Operations

Now we describe the system operations of our proposed architecture using the steps shown in Figure 5.2. 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 key dk for the user or sk for the role, and sends the secret key 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. Step 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. 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, she or 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 $g^{p_{i,\mathcal{M}}(s)}$, $g^{p_{k,\mathcal{N}}(s)}$ when the role hierarchy or user membership has changed. More specifically, $g^{p_{i,\mathcal{M}}(s)}$ is re-computed if the role parameter A_R has changed since the last decryption request from the same user, and $g^{p_{k,\mathcal{N}}(s)}$ 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 certain data, the user will need to find its index first. To 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 [27, 60, 25]. Using one such searchable encryption scheme, a user can make a query to the public cloud about the data that she or 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, she or he can simply use the secret decryption key to sign the data using the IBS scheme described in section 5.2.3.

5.4 Implementation

5.4.1 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[79] web services, and the web services are hosted in Apache Tomcat. The clouds use the HyperSQL[75] database which can be easily replaced by other databases for server side data storage. The client side is written as a 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 [124]. In our implementation, we use a 163-bit MNT curve [103] with the embedding degree of 6. In practice, the security of a 160-bit elliptic curve is roughly equivalent to 1024-bit RSA[8]. 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[85] as the symmetric encryption algorithm *Enc*. The reason for this choice is that a stream cipher can work with the MTOM[129] 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}[111] for the best known attack to ISAAC [6].

To protect the data integrity while using a stream cipher, we deployed the web services through a SSL channel. When the encrypted data is stored in the cloud, it is stored along with a signature generated by the data owner. Since the symmetric

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

Table 5.1: RBE System Benchmark - Ciphertext Size (Bytes)

encryption algorithm can be chosen independently from the RBE scheme, it can be replaced by other symmetric encryption algorithms in other implementations depending on the system's requirements, such as using a block cipher algorithm.

We use jPBC[34] and PBC[96] as our pairing-based crypto library (jPBC wraps the PBC library to generate a MNT curve), and Bouncy Castle crypto library[28] for SHA-1 and ISAAC.

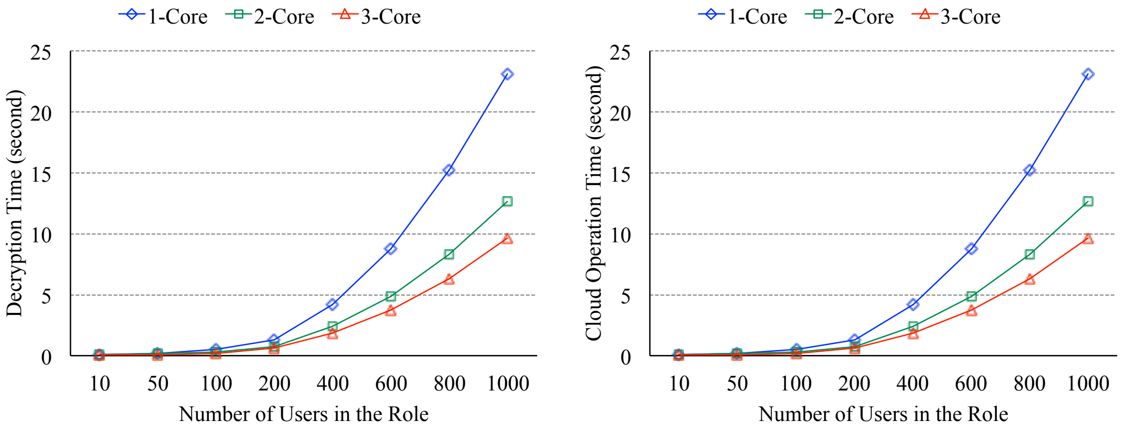
5.4.2 Experimental Evaluation

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 5.1 shows the ciphertext sizes when the sizes of plaintext are 1000 Bytes, 10000 Bytes and 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 a 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



(a) Public Cloud Decryption Time

(b) Public Cloud Operation Time in User Management

Figure 5.3: RBE System Benchmark - Public Cloud Operation Time

48 bytes, which is convenient for users. A non-constant size decryption key will usually make it difficult for users to decide the memory requirements that are needed on the client devices to store the keys. Our system does not have this problem.

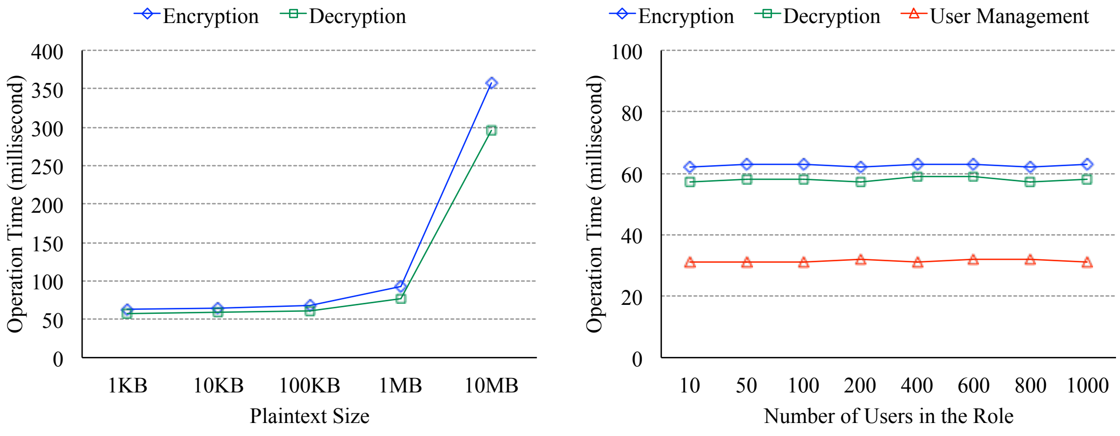
Encryption and decryption are the most frequently used operations in the system. Since we have a split decryption algorithm to run it in both client and the cloud, we first measured the time taken at the cloud for performing decryption. The time for cloud decryption is measured from the time the public 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 used one thread as the master thread in the computation task, the maximum cores that we have simulated on our quad-core server is three.

Figure 5.3(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, reducing 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. Figure 5.3(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. Figure 5.4(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 was successful. The decryption time was measured from the time when a user starts receiving the ciphertext from the cloud until the time the plaintext is saved



(a) Different Plaintext Size

(b) Different Number of Users

Figure 5.4: RBE System Benchmark - Client Operation Time

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 Figure 5.4(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 operation time of clients is reduced dramatically. Figure 5.4(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.

5.5 Conclusion

In this chapter, we first proposed a new RBE scheme that achieves efficient user revocation. We then 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. 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.

6

Administrative Model for Role-based Encryption

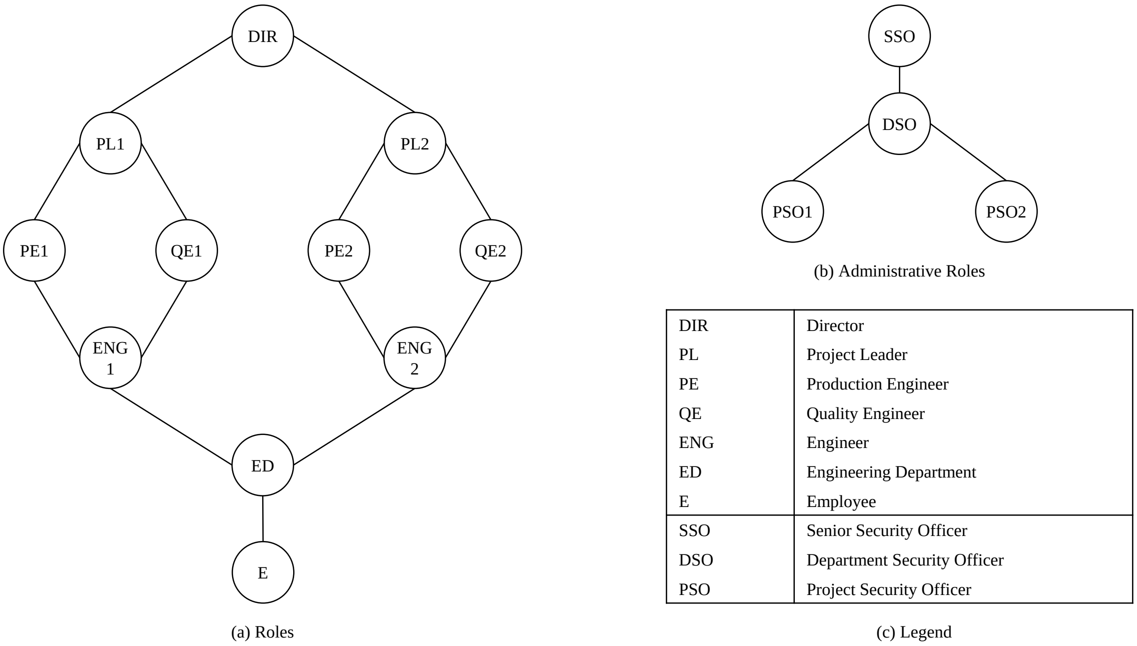
In this chapter, we propose a cryptographic administrative model AdC-RBAC to manage and enforce role-based access policies for RBE schemes in large-scale cloud systems. The AdC-RBAC model uses cryptographic techniques to ensure that the administrative tasks such as user, permission, and role management are performed only by authorised administrative roles. Any other party, including the cloud providers themselves, cannot change RBAC systems and policies. Our proposed model uses RBE techniques to ensure that only administrators who have the permissions to manage a role can add or revoke users to or from the role, and owners can verify whether a role is managed by qualified administrators before giving out their data. We show how the proposed model can be used in an untrusted cloud while guaranteeing its security using cryptographic and trusted access control enforcement techniques. We describe three components in this model: UAM for user membership management, PAM for permission management, and RAM for role management. We describe the operation of the AdC-RBAC model using a proposed variant RBE scheme based on the RBE scheme described in Chapter 5.2.1. We describe how our proposed AdC-RBAC model can be integrated with the variant RBE scheme.

The rest of this chapter is organised as follows. Section 6.1 introduces the background of our work. In section 6.2, we propose the new variant of RBE scheme which supports decentralised role management. In section 6.3, we present the cryptographic administrative RBAC model AdC-RBAC, and describe how the administration tasks of the RBE scheme can be simplified in a large-scale system, and how the administrative policies of the RBAC system are enforced in a cloud system. Finally, section 6.4 concludes the chapter.

6.1 Administration in Role-based Access Control

RBAC has been widely used for security administration in distributed systems since being first formalised in the 1990's. However, the administration of RBAC systems themselves has been less widely studied. In small RBAC systems, a central authority is usually sufficient to manage all the users and permissions. However large-scale RBAC systems may have hundreds or even thousands of roles and hundreds of thousands of users and permissions. In such cases, it becomes impractical to centralise the task of managing these users and permissions, and their relationships with the roles to a small team of security administrators. Therefore, decentralising the administration tasks of RBAC systems is an important issue when developing such large-scale RBAC systems.

Several administrative RBAC (ARBAC) models have been developed to provide solutions to decentralise the administration privileges. The administrative model for RBAC was first considered in [115], and a comprehensive model was proposed in this work, called ARBAC97. It was later extended and improved in [119, 108, 109, 44, 43, 42]. A common feature of these works is managing a RBAC system using RBAC



(a) Roles

(b) Administrative Roles

DIR	Director
PL	Project Leader
PE	Production Engineer
QE	Quality Engineer
ENG	Engineer
ED	Engineering Department
E	Employee
SSO	Senior Security Officer
DSO	Department Security Officer
PSO	Project Security Officer

(c) Legend

Figure 6.1: ARBAC97 Role Hierarchy Example

itself. The administration privileges are decentralised to a set of administrative roles in these models, and administrative policies are specified to limit the privileges of administrative roles. Each administrative role is assigned an administration domain, and the role is allowed to perform administration tasks only on the roles that reside in the administration domain. Next we review two existing administrative models for RBAC systems.

6.1.1 ARBAC97 Model

The challenge of administering RBAC was first considered in [115] where a comprehensive administrative model, ARBAC97, was introduced. ARBAC97 is based on the RBAC model defined in [120]. In ARBAC97, a RBAC system is administered by an administrative RBAC, which contains a set of administrative roles AR that are separate from the roles R in the normal RBAC system.

The ARBAC97 model defines three components: URA97 for user-role assignment, PRA97 for permission-role assignment, and RRA97 for role-role assignment. In each component, the ability to perform the assignment is associated with administrative roles. In other words, the administrative roles have control over the administrative operations in the normal RBAC. For each administrative role in these components, the authority range of the role is specified by a concept of *role range*, which is defined as a set of roles in the normal RBAC, and is denoted by following notations

$$[x,y] = \{r \in R \mid x \leq r \wedge r \leq y\} \tag{6.1}$$

$$(x,y] = \{r \in R \mid x < r \wedge r \leq y\} \tag{6.2}$$

$$[x,y) = \{r \in R \mid x \leq r \wedge r < y\} \tag{6.3}$$

$$(x,y) = \{r \in R \mid x < r \wedge r < y\} \tag{6.4}$$

where $x < y$ means that role y inherits the permissions from role x.

The URA97 component defined the following two relations: *can-assign:* $(a, \mathcal{C}, \mathcal{R})$ and *can-revoke:* $(a, \mathcal{R})$, where a is an administrative role, and $\mathcal{R}$ is either a role range or a role set that has been explicitly specified. $\mathcal{C}$ denotes a prerequisite condition which is a boolean expression using the operators $\neg$, $\vee$, $\wedge$ on regular roles of the system. Sandh *et al.* [115] gives the example role hierarchies as shown in Figure 6.1 to illustrate the model. In this example, *can-assign*$(DSO, ED \wedge \neg PL1, (ENG2, PL2])$ means that the members of the administrative role DSO can assign a user, who is currently a member of the role ED and not a member of the role $PL1$, to be a member of regular roles of the set $\{PL2, PE2, QE2\}$. *can-revoke*$(PSO2, [ENG2, PL2])$ means that the members of the administrative role $PSO2$ can revoke membership of a user from any regular role in the set $\{PL2, PE2, QE2, ENG2\}$. PRA97 is defined as the dual of the URA97 model. Thus it has two relations *can-assignp:* $(a, \mathcal{C}, \mathcal{R})$ and *can-revokep:* $(a, \mathcal{R})$ which are similar to the *can-assign* and *can-revoke* in the URA97.

The RRA97 component defined five relations: *can-assigna*, *can-revokea*, *can-assigng*, *can-revokeg*, *can-modify*. In order to support establishing relationships among different administrative models, two special types of roles are defined in RRA97: *Abilities* are roles that can only have permissions and other abilities as members, and *Groups* are roles that can only have users and other groups as members. The relations *can-assigna*, *can-revokea* are defined for *Abilities* role assignment and revocation, and the relations *can-assigng*, *can-revokeg* are defined for *Groups* role assignment and revocation. In this chapter, we do not consider these four relations as they are not required by RBE schemes.

The relation *can-modify* specifies the authorisation of operations role creation, deletion, and modification, and is defined as $(a, \mathcal{R})$ where a is an administrative role, and $\mathcal{R}$ is a role range. The role range in RRA97 is a special case where no end points is included, and any role range referenced in the *can-modify* relation is called an *authority range*. For example, *can-modify:* $(PSO1, (ENG1, PL1))$ means that the members of the administrative role $PSO1$ can create, delete, and modify roles in the range $(ENG1, PL1)$. In order to maintain global consistency when modifying roles, several restrictions have been placed for defining the *authority range* in RRA97. First, authority ranges do not overlap partially. In the example, as the role hierarchies, $(E2, DIR)$ and $(ED, PL1)$ are partially overlapping, they are not allowed to be defined at the same time. Secondly, authority ranges must be encapsulated. ARBAC97 defines the encapsulated range as follows.

Definition 6.1 *A range (x, y) is said to be encapsulated if for all roles $r_i \in (x, y)$ and all roles $r_e \notin (x, y)$,*

$$r_e > r_i \Leftrightarrow r_e > y, \quad \textit{and} \quad r_e < r_i \Leftrightarrow r_e < x$$

The ARABC97 model further defined the restrictions on individual operations in the relation *can-modify*, including role creation, role deletion, edge insertion, and edge deletion. ARBAC97 has been extended in ARBAC99[119] and ARBAC02[108] where there have been changes to the URA and PRA relations.

6.1.2 SARBAC Model

Crampton and Loizou [44, 43] proposed another administrative model, SARBAC, for RBAC, and the model was then improved in [42] where a role-based administration template (RBAT) model was proposed. The intention of SARBAC is to improve the

RRA model in ARBAC97. RRA97 was based on encapsulated ranges, which are relatively complicated to administer and require a significant effort in deciding an administrative policy. In order to simplify the RRA model, SARBAC introduced the concept of the administrative scope, which is defined based on the role hierarchies. Similar to the authority range in ARBAC97, an administrative scope is used to specify a set of roles that can be modified by an administrative role. In this chapter, we follow the description of SARBAC given in [42].

First, let us look at the notations given in [42]. Let $s \in R$; define $\uparrow s = \{r \in R : r \geq s\}$ and $\downarrow s = \{r \in R : r \leq s\}$. The expression $\uparrow s \cup \downarrow s$ is denoted as $\updownarrow s$. The administrative scope is defined as follows.

Definition 6.2 *The administrative scope of a role r, denoted by $\sigma(r)$, is defined to be*

$$\sigma(r) = \{s \ \in \downarrow r : \uparrow s \ \subseteq \updownarrow r\}$$

Based on this definition, a role-based administration template (RBAT) was introduced. RBAT defines a single relation *can-administer:* (a, r), which means that an administrative role a can control the set of roles in $\sigma(r)$. For example, *can-administer*$(PSO1, PL1)$ means that the administrative role $PSO1$ can create, delete, and modify the roles in $\sigma(PL1)$, which specifies the set of roles $\{PL1, PE1, QE1, ENG1\}$.

One feature of SARBAC is that when some RRA operations affect the authority ranges of the administrative roles, no new relation policy needs to be specified; the administrative scope changes following the updating of the role hierarchy, while in ARBAC97, these operations are not allowed. RBAT formalised the relations between the RRA operations and the administrative scopes by defining the scope preserving hierarchy operations and the preserving conditions of the scopes.

6.2 A RBE Scheme for Decentralised Role Management

In this section, we propose our variant RBE scheme based on the RBE scheme described in section 5.2.1. This variant RBE scheme will be later used with our proposed cryptographic administrative model AdC-RBAC.

6.2.1 Formulation of the RBE Scheme

The RBE schemes defined in previous chapters allow the user membership to be managed by individual roles which makes the scheme easy to be used with administrative RBAC systems as the user management in the schemes has already been decentralised. To use RBE schemes with ARBAC, the administration tasks in the RBE schemes also need to be decentralised so that they can be performed by the administrative roles instead of the system administrator. Moreover, in ARBAC models, one single administrative role can administer multiple roles, so each individual role does not have to be allocated a role manager. Therefore, we replace the party role managers RM with administrative roles AR in the new RBE scheme. Taking into account the above requirement changes, we modify the definition of the RBE scheme in Chapter 3, and the algorithms of the new RBE scheme are described as follows,

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.

CreateRole $(\mathsf{mk}, \mathsf{ID}_R)$: an algorithm executed by the SA to create a new role with identity ID_R. The algorithm returns the role secret sk_R, a set of public parameters pub_R for the role and an empty user list $\mathcal{RUL}$ which will list all the users who are the members of that role.

CreateUser $(\mathsf{mk}, \mathsf{ID}_U)$: an algorithm executed by the SA to add a new user to the system. ID_U is the unique identity of the new user. The algorithm returns the secret decryption key dk, which the user will use to decrypt all the data retrieved from the system.

ManageRole $(\mathsf{pk}, \mathsf{ID}_R, \mathcal{T})$ is executed by the AR to manage a role with the identity ID_R in the role hierarchy $\mathcal{T}$. This operation publishes a set of public parameters pub_R to the cloud.

AddUser $(\mathsf{pk}, \mathsf{sk}_R, \mathcal{RUL}_R, \mathsf{ID}_U)$: an algorithm executed by the administrative role AR of a role, taking as input the role secret sk_R, to add the user ID_U to the set of users who are members of that role by updating the role's public parameter pub_R and the user list $\mathcal{RUL}_R$ if the user qualifies the role.

RevokeUser $(\mathsf{pk}, \mathsf{sk}_R, \mathcal{RUL}_R, \mathsf{ID}_U)$: an algorithm executed by the administrative role AR on input an identity ID_U of the user U and the role secret sk_R, removes U from the $\mathcal{RUL}$ and updates the role's public parameters.

Encrypt $(\mathsf{pk}, \mathsf{pub}_R, M)$: an algorithm executed by the owner to encrypt the private data M, outputs the ciphertext C.

Decrypt $(\mathsf{pk}, \mathsf{pub}_R, \mathsf{dk}, C)$: an algorithm executed by the user to decrypt the ciphertext C, taking as input the user decryption key sk_R, outputs the plain text message M if the user has the permission to access the data, and fails otherwise.

In this RBE definition, all the administration operations for the RBAC are carried out by the administrative roles AR, and the system administrator SA only needs to participant in the system setup and the key generation phase. After the role and user secret keys are generated in the algorithms *CreateRole* and *CreateUser*, they are sent to the administrative roles and the users respectively via a secure channel. Note that in the above it has been assumed that the cloud provider has the capability to store both the ciphertexts and the public parameters of the system. The latter includes the public parameters for all the roles and the public key, pk, of the system. We assume that the information that is private to each user, such as the decryption key of each user, is held securely by the user. We also assume that the system and the role secret information are stored securely by the SA and role managers respectively. We will refer to some of the above mentioned operations and parameters in the following sections.

6.2.2 The RBE Scheme Construction

Now we describe the RBE scheme which is designed using asymmetric bilinear groups as follows,

Setup: Generate three groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, and a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. 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 $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$, $H_2 : \mathbb{G}_T \rightarrow \mathbb{G}_1$ and a secure symmetric encryption scheme Enc whose key space is $\mathbb{G}_T$. The master secret key mk and system public key pk are defined as

$$\mathsf{mk} = (s, k, g), \quad \mathsf{pk} = (w, v, g^k, h, h^s, \cdots, h^{s^{q_1}}, h^k, h^{k \cdot s}, \cdots, h^{k \cdot s^{q_2}})$$

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

and q_1 is the maximum number of users that each role can have and q_2 is the maximum depth of the role hierarchy. If q_1 is less than q_2 in practice, we set q_1 equal to q_2.

CreateRole(mk, ID_R): To create a role R with identity ID_R, SA computes the role secret as

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

and gives sk_R to the administrative role of R along with the $\mathcal{RUL}_R$ which is initially set to empty. This gives the administrative role the ability to manage the user membership of the role.

CreateUser(mk, ID_U): To create a user U with identity ID_U in the system, the SA computes the user decryption key as

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

and gives dk_U to the user U. This effectively adds the user to the system. The user will use dk_U as their decryption key in recovering information from the system.

ManageRole(pk, ID_R, $\mathcal{T}$)**:** 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 ancestor roles of a role R with the identity ID_R. To place this role R in the role hierarchy, AR computes and publishes the tuple $(A_R, B_R, \mathcal{RUL}_R)$ as role public parameters in the cloud where

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

AddUser(pk, sk_{R_i}, $\mathcal{RUL}_{R_i}$, ID_{U_k})**:** Assume that the administrative role AR of the role R_i wants to add a user U_k with the identity ID_{U_k} as a member of the role R_i. $\mathcal{RUL}_{R_i}$ is the set of n users who are the members of the role R_i and U_k is not in $\mathcal{RUL}_{R_i}$. The administrative role AR chooses two random values $r_i, t_i \leftarrow \mathbb{Z}_p^*$ if $\mathcal{RUL}_{R_i}$ is empty, or uses the existing r_i, t_i otherwise. Then it computes

$$K_i = v^{r_i}, \quad T_i = g^{-t_i}, \quad W_i = w^{-r_i}, \quad V_i = h^{r_i\cdot(s+H_1(\mathsf{ID}_{U_k}))\prod_{j=1}^{n}(s+H_1(\mathsf{ID}_{U_j}))}$$

and

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

AR adds ID_{U_k} into $\mathcal{RUL}_{R_i}$, and sends the tuple (T_i, W_i, V_i, S_i) to the cloud via a secure channel. 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 administrative role AR first removes ID_{U_k} from role user list $\mathcal{RUL}_{R_i}$. Then AR chooses two random values $r_i', t_i' \leftarrow \mathbb{Z}_p^*$ and re-computes

$$K_i' = v^{r_i'}, \quad T_i' = g^{-t_i'}, \quad W_i' = w^{-r_i'}, \quad V_i' = h^{r_i'\cdot\prod_{j=1,j\neq k}^{n}(s+H_1(\mathsf{ID}_{U_j}))}$$

and

$$S'_i = H_2(K'_i) \cdot \mathsf{sk}_{R_i} \cdot g^{kt'_i} = H_2(v^{r'_i}) \cdot g^{\frac{1}{s+H_1(\mathsf{ID}_{R_i})}+kt'_i}$$

AR sends the new role parameters (T'_i, W'_i, V'_i, S'_i) to the cloud, and the cloud replaces the public parameters of the role with

$$(\mathsf{ID}_{R_i},\ A_i,\ B_i,\ W'_i,\ V'_i,\ S'_i,\ \mathcal{RUL}_{R_i})$$

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 the encryption scheme Enc to encrypt the message M with the key K, and uploads the ciphertext $Enc_K(M)$ 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 and returns the following tuple to the user U_k,

$$\{C_1, C_2, D, Enc_K(M), \quad \text{where} \quad D = \hat{e}(T_i, C_3)\}$$

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

$$K' = (\hat{e}(C_1, h^{p_{i,\mathcal{M}}(s)}) \cdot \hat{e}(\frac{S_i}{H_2(K_i)}, C_2) \cdot D)^{\frac{1}{\prod_{j=1, j\neq i}^{m} H_1(\mathsf{ID}_{R_j})}}$$

where

$$K_i = (\hat{e}(\mathsf{dk}_{U_k}, V_i) \cdot \hat{e}(W_i, h^{p_{k,\mathcal{N}}(s)}))^{\frac{1}{\prod_{j=1, j\neq k}^{n} H_1(\mathsf{ID}_{U_j})}}$$

and

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

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

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

6.2.3 Security Proof

In this subsection, we only prove CPA security of our RBE scheme based on GDDHE assumption. Similar to previous RBE constructions, our RBE scheme can be extended to a CCA-secure scheme using some exist generic conversion methods that can convert a CPA-secure scheme to a CCA-secure scheme.

Theorem 6.1 *The proposed RBE scheme is chosen plaintext secure under the GDDHE assumption.*

To prove Theorem 6.1, we start by defining another specific GDDHE problem where it shows that $f = zh(s)$ is independent from P and Q. In the following definition, s, k, z, c are random elements chosen from $\mathbb{Z}_p^*$ as described in our RBE scheme.

Definition 6.3 GDDHE Problem III *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 $v = \hat{e}(g, h)$. Let f and g be two coprime polynomials with pairwise distinct roots, of respective orders m and n. Given $K \in \mathbb{G}_T$ and*

$$g_0, g_0^s, \ldots, g_0^{s^{m-1}},\ g_0^{s\cdot c\cdot g(s)},\ g_0^{k\cdot c\cdot g(s)},\ g_0^{z\cdot s\cdot c\cdot g(s)}$$

$$h_0, h_0^s, \ldots, h_0^{s^n}, h_0^{f(s)}, h_0^{s\cdot f(s)}, h_0^{h(s)}, h_0^{k\cdot h(s)}, h_0^{z\cdot h(s)}, h_0^{k\cdot z\cdot h(s)}$$

in deciding whether K is equal to $\hat{e}(g_0, h_0)^{z\cdot c\cdot h(s)}$ 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 the 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 6.3. Next we define the following polynomials:

- $g(x) = \prod_{i=1}^{m}(x + H_1(\mathsf{ID}_{R_i}))$
- $f(x) = \prod_{i=1}^{n}(x + H_1(\mathsf{ID}_{U_i}))$
- $h(x) = g(x) \cdot f(x)$
- $g_i(x) = \frac{c \cdot g(x)}{x + H_1(\mathsf{ID}^*_{R_i})}$, 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 by randomly choosing s, k, z, t, c from $\mathbb{Z}_p^*$.

$$h = h_0^{f(s)}, \; w = g_0^{s \cdot c \cdot g(s)}, \; v = \hat{e}(g, h) = \hat{e}(g_0, h_0)^{c \cdot h(s)}$$

This implies that $g = g_0^{c \cdot g(s)}$. Then $\mathcal{B}$ defines the public key as $\mathsf{pk} = (w, v, g^k, h, h^s, \cdots, h^{s^q}, h^k, h^{k \cdot s}, h^{k \cdot s^q})$, 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 $q_1, \ldots, q_k$,

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

$$\mathsf{sk}_{R_i} = g_0^{g_i(s)} = g^{\frac{1}{s+H_1(\mathsf{ID}^*_{R_i})}}$$

- *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

$$W = g_0^{-r'sc\cdot g(s)} = w^{-r'},$$

$$V = h_0^{r'\cdot f(s)\cdot f(s)} = h^{r' f(s)} = h^{r'\cdot\prod_{i=1}^{n}(s+H_1(\mathsf{ID}^*_{U_i}))}$$

$$S = H_2(\hat{e}(g_0, h_0)^{r'c\cdot h(s)}) \cdot g_0^{g_i(s)} \cdot g_0^{kt'c\cdot g(s)}$$

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.

$$W = g_0^{-rsc\cdot g(s)} = w^{-r},\ V = h_0^{r\cdot f(s)} = h^r$$

$$S = H_2(\hat{e}(g_0, h_0)^{rc\cdot h(s)}) \cdot g_0^{g_i(s)} \cdot g_0^{ktc\cdot g(s)}$$

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,

$$C_1 = g_0^{-zsc\cdot g(s)} = w^{-z},\ C_2 = h_0^{zh(s)} = A^z$$

$$C_3 = h_0^{kz\cdot h(s)},\ D = \hat{e}(g_0, h_0)^{-tc\cdot g(s)\cdot kz\cdot h(s)},\ K = K'$$

Note that if $K' = \hat{e}(g_0, h_0)^{zc \cdot h(s)}$, 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 *CreateRole* 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

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

6.2.4 Discussion

One feature of the variant RBE scheme is that all the administrative tasks, such as role management, permission and user assignment, are decentralised. The system administrator does not need to be involved in any administrative tasks. Since no secret parameter is required to construct the role hierarchy, any party who has the access to the system public keys can define the public parameters of a role. In the administrative RBAC model that we will propose, we will show how to restrict the role management permissions in the RBE scheme. For the user management, a role secret key is generated for each role, and the role secret key can be given to any administrative role to manage the user membership. Note that this does not manage the permissions of the roles, only the users. Therefore, the problems of managing the user-role mapping are transferred into the problems of managing role secret keys.

Any data owner is allowed to encrypt data using the role public parameters and thereby add a permission to the role. While we recognise that this is unusual for RBAC systems, it does allow significant flexibility in adding data to the cloud storage. This activity could be controlled by restricting access to the role public parameters, or at least the part required for encryption. At the extreme, we could restrict access to the encryption-related role public parameters to the administrative role itself, restoring that entity to control over the role-permission mapping. Data owners would then make requests of the administrative role for it to encrypt data to the role. However, this would complicate the explanation of the algorithms corresponding to the operations set out above.

In this RBE scheme, only users that are members of the role to which the data was encrypted, or members of one of the ancestor roles of that role, can decrypt the ciphertext. It is assumed that users keep secret the decryption key given to them by the SA. It is further assumed that users have some credentials that they can use to prove their identity to administrative roles, and the administrative roles will use these credentials as the basis for deciding whether or not to assign users to the roles that they manage. A user is able to join a role after the owner has encrypted the data for that role, and the user will be able to access that data from then on, without the owner needing to re-encrypt the data. A user can be revoked at any time (e.g. if the user is found to be malicious), in which case the revoked user will not have access to any data encrypted to this role.

So far we have only discussed the read permission; that is, a user who belongs to a role to which a message was encrypted can decrypt and read the message. Now let us consider the write permission. The write permission is for encrypting data to a role, and we have discussed previously that we can either allow any user to encrypt messages or restrict the write access by giving out the role public parameters to the authorised users only. Now let us extend this case to a more general scenario. Assume an owner wishes to assign multiple permissions, such as read, add, modify, and delete, to a role, and these permissions are protected by cryptographic keys. For example, the read and the add permissions can be controlled by managing the associated keys as we discussed above, and the system can require the modification and the deletion operations to come with signatures generated by separate keys. The owner can create a key for each different permission, and encrypts these keys to this role. Since the user who has access to messages encrypted to this role can recover the keys for the permissions of that role, the user is able to obtain these permissions and receives the ability to carry them out.

6.3 Administrative Model for RBE Schemes

In this section, we propose a cryptographic administrative model AdC-RBAC that can manage and enforce role-based access policies for RBE schemes in large-scale cloud systems, and we use the role hierarchies shown in Figure 6.1 as an example to explain our proposed administrative model. The AdC-RBAC model uses cryptographic techniques to ensure that the administrative tasks such as user, permission and role management are performed only by authorised administrative roles. This model consists of three components: UAM for user membership management, PAM for permission management, and RAM for role management.

6.3.1 User Administration Model

In RBE schemes, user membership administration has already been decentralised. The management of user membership for each role is controlled by a role secret key sk_R which is used as an input parameter to the *AddUser* and *RevokeUser* operations. Only the parties who hold this key can add or revoke users to or from this role. Therefore, we define the following relation for UAM.

Definition 6.4 *User management policies in AdC-RBAC are specified by the following relation*

$$\text{can-manage} \subseteq AR \times \mathcal{R}$$

where AR is the set of administrative roles and $\mathcal{R}$ is either a role range or a role set that has been explicitly specified.

The example shown in Table 6.1(a) means that the members of the administrative role $PSO1$ can manage the user membership of the roles specified by the range $[ENG1, PL1]$. To enforce this relation in UAM, the secret keys sk_R of the roles that

Administrative Role	Role Range
PSO1	[ENG1, PL1]

(a) Single relation

Administrative Role	Role Range
DSO PSO1 PSO2	[DIR, DIR] [ENG1, PL1] [ENG2, PL2]

(b) Multiple relations

Table 6.1: AdC-RBAC Model: can-manage Example

are within the range are encrypted to the role $PSO1$. We note that the problem of specifying the role range has been transformed into the problem of encrypting a set of role secret keys. The concepts of partial overlap and incomparable have been introduced in [115] to define restrictions for the ARBAC97 model. We give our definitions as follows.

Definition 6.5 *Assume that there exist two key sets K_1 and K_2 which correspond to two role ranges $\mathcal{R}_1$ and $\mathcal{R}_2$. We say two ranges overlap partially if $K_1 \cap K_2 \neq \emptyset$ and $K_1 \nsubseteq K_2$ and $K_2 \nsubseteq K_1$. Ranges $\mathcal{R}_1$ and $\mathcal{R}_2$ are said to be incomparable if $K_1 \cap K_2 = \emptyset$.*

In UAM, to avoid any potential conflict that may be caused by managing the user membership of a role by multiple administrative roles, we introduce the following restriction.

Definition 6.6 *In UAM, role ranges are incomparable.*

In some cases, the system may want an administrative role to have full control over another administrative role; that is, one role range is allowed to be the superset of another role range. In UAM, this can be achieved by the inheritance in the encryption. Table 6.1(b) shows a more complex scenario which contains a list of relations. We explain using this example to show how we can achieve this requirement.

First, we assume that the administrative roles are organised in a hierarchy as shown in Figure 6.1(b), and the role DSO inherits permissions from the role $PSO1$ and $PSO2$.

We use our RBE scheme to encrypt the corresponding set of role secret keys to each administrative role. Assume that the secret keys for the roles are encrypted to the administrative roles following the relations specified in Table 6.1(b). Since the role DSO can decrypt the data encrypted to the role $PSO1$ and $PSO2$, it can recover the secret keys for roles in the set $[DIR, DIR] \cap [ENG1, PL1] \cap [ENG2, PL2]$; hence it can manage the user membership of these roles. Therefore the effective range that the role DSO can manage is $(ED, DIR]$.

6.3.2 Permission Administration Model

Recall that in the RBE system, all the owners who can access the system are able to encrypt the data to the roles; hence in a general case, anyone is allowed to encrypt the data to the roles in a RBAC system. Since the RBE scheme does not have specific requirements on permissions-role assignment, there is no restriction on the permission assignment in the RBE scheme. Therefore, the permission administration model is not mandatory in the AdC-RBAC model.

However, in some cases, the RBAC system may want to allow only the specific administrative roles to encrypt data to roles. For example, in our proposed RBE scheme, role public parameters pub_R are required in encrypting a data to a role. These role parameters are defined as public, so that the data can be encrypted to the roles by any parties using these parameters as part of the encryption key. In order to restrict permission assignment in PAM, we encrypt these role parameters to the administrative roles who are permitted to assign permissions to these roles, so that only the authorised administrative roles can encrypt data to these roles. Hence we define the following relations in PAM.

Definition 6.7 *Permission assignment policies in AdC-RBAC are specified by the following relation*

$$\text{can-assign} \subseteq AR \times \mathcal{R}$$

where AR is the set of administrative roles and $\mathcal{R}$ is either a role range or a role set that has been explicitly specified.

Administrative Role	Role Range
PSO1	[E, PL1]

(a) Single relation

Administrative Role	Role Range
DSO	[DIR, DIR]
PSO1	[E, PL1]
PSO2	[ED, PL2]

(b) Multiple relations

Table 6.2: AdC-RBAC Model: can-assign Example

The example shown in Table 6.2(a) means that the members of the administrative role $PSO1$ can assign permissions to the roles specified by the range $[E, PL1]$. To enforce this relation in PAM, the parameters pub_R of the roles within the range are encrypted to the role $PSO1$.

Table 6.2(b) shows an example with multiple relations. Different from UAM, one administrative role assigning permissions to a normal role does not affect permissions assignment of other administrative roles. Therefore we do not have restrictions in PAM, and ranges specified are allowed to overlap with each other. We discuss two different modes using this example for our PAM: *flat mode* and *hierarchy mode*.

Flat mode

In a *flat mode*, the administrative roles are organised in a flat manner; that is, these roles do not inherit permissions from each other. Each administrative role is only allowed to assign permissions to the roles that are specified in the role range in the relation. In the example, DSO can only assign permissions to the role DIR, and the role $PSO1$, $PSO2$ can only assign permissions to the roles in the ranges $[ENG1, PL1]$ and $[ENG2, PL2]$ respectively. To encrypt the roles' parameters (which are used to encrypt data) to the administrative roles, an identity-based encryption (IBE) scheme [41, 21, 18, 19] is sufficient. Administrative roles can use their private keys to decrypt the parameters pub_R of the roles for which they have the authority to assign permissions; hence they can encrypt the data using the parameters of the corresponding role.

Hierarchy mode

In a *hierarchy mode*, the administrative roles are organised in a hierarchical manner; that is, these roles can inherit permissions from other roles. As shown in Figure 6.1(b), the role DSO inherits permissions from the role $PSO1$ and $PSO2$. Hence the difference from the *flat mode* is that the role DSO can assign permissions not only to the role DIR, but also to all the roles to which the role $PSO1$ and $PSO2$ can assign permissions. The IBE scheme cannot be used to encrypt the role parameters in this scenario as it cannot reflect the relationship among these administrative roles. Thus we need to use the RBE scheme for the encryption in this mode. Assume that the parameters for the regular roles are encrypted to the administrative roles following the relations specified in Table 6.2(b). Since the role DSO can decrypt the data encrypted to the role $PSO1$ and $PSO2$, it can recover the parameters of all the roles in the range $[E, DIR]$; hence it can assign permissions to all these roles.

6.3.3 Role Administration Model

In the proposed RBE scheme, we discussed that it is possible for any administrative role to administer the regular roles. Now we see how to restrict the role management operations in the RBE scheme. In RAM, we use authentication mechanisms to allow other parties to verify whether or not a role in the RBAC system is created/modified by an authorised administrative role. First we define the following relation.

Definition 6.8 *Role administration policies in AdC-RBAC are specified by the following relation*

$$\text{can-administer} \subseteq AR \times \mathcal{R}$$

where AR is the set of administrative roles and $\mathcal{R}$ is either a role range or a role set that has been explicitly specified.

We follow the ARBAC97 and require the same restriction on authority ranges in this relation.

Administrative Role	Role Range
DSO	(E, DIR)
PSO1	(ENG1, PL1)
PSO2	(ENG2, PL2)

Table 6.3: AdC-RBAC Model: can-administer Example

Definition 6.9 *In RAM, authority ranges of the administrative roles do not overlap partially and must be encapsulated, and the edge roles of the authority ranges cannot be modified.*

In RAM, we use identity-based signature schemes [71, 35, 10] to certify the authority of the administrative roles. We use $\mathcal{D}_{AR}(M)$ to denote data M's signature which is signed to the identity of the administrative role AR. In order to facilitate the verification of the authority, we define a set of administrative parameters AP for each regular role. Now consider the relation *can-administer:* $(a, (x, y))$ which means that the members of an administrative role a can administer the roles in the range (x, y). In this relation, the following parameters are associated with the edge roles of the range,

$$\langle \ \mathsf{ID}_r, \mathcal{PR}_r, \mathcal{SR}_r, AR_r, \tau, \mathcal{D}_s(P_r) \ \rangle$$

where P_r denotes $\mathsf{ID}_r||\mathcal{PR}_r||\mathcal{SR}_r||AR_r||\tau$, ID_r is the identity of the edge role of the range, $\mathcal{PR}_r$ is the set of the identities of r's immediate senior roles, $\mathcal{SR}_r$ is the set of the identities of r's immediate junior roles, AR_r is the identity of the administrative role who can administer the range, τ denotes the type of the edge: upper bound or lower bound, and $\mathcal{D}_s(P_r)$ is the signature on the parameters, which is issued by the most senior administrative role who defines the relations in RAM. These parameters are computed and attached to the role when this role is specified as the edge of the authority range by the most senior role s. For other regular roles within the range, the parameters associated with them are as follows,

$$\langle \ \mathsf{ID}_r, \mathcal{PR}_r, \mathcal{SR}_r, AR_r, \mathcal{D}_a(P_r) \ \rangle$$

Algorithm 1 Algorithm to Verify the Authorisation

```
function IsEdge($r, a, t$)
    if role $r$ has parameter $\tau$ and $\tau = t$ and $AR_r = \mathsf{ID}_a$ then
        return $true$
    end if
    return $false$
end function

function VerifyRole($r, a, t$)
    if role $r$ has parameter $\tau$ then
        return verify($\mathcal{D}_s(P_r), \mathsf{ID}_s$)
    end if
    return verify($\mathcal{D}_a(P_r), \mathsf{ID}_a$)
end function

function VerifyAuth($r, a, t$)
    $State_r \leftarrow false$
    if IsEdge($r, a, t$) then
        $State_r \leftarrow$ verify($\mathcal{D}_s(P_r), \mathsf{ID}_s$)
    else if VerifyRole($r, a, t$) then
        if $t =$ upper-bound then
            $\mathcal{R} \leftarrow \mathcal{PR}_r$
        else if $t =$ lower-bound then
            $\mathcal{R} \leftarrow \mathcal{SR}_r$
        end if
        for each $x \in \mathcal{R}$ do
            $State_r \leftarrow State_x$
            if $State_r = false$ then  ▷ Only verify the role that has not been verified
                if $a \geq AR_x$ then        ▷ Check if $a$ inherits the permissions from $AR_x$
                    $State_r \leftarrow$ VerifyAuth($x, a, t$)
                end if
                if $State_r = false$ then
                    return $State_r$
                end if
            end if
        end for
    end if
    return $State_r$
end function

Require: Initiate a $State_r$ for each role $r \in R$ to $false$
    Input $w$  ▷ Checking if the role $w$ is administered by the authorised administrator.
    $b \leftarrow false$
    if VerifyAuth($w$, $AR_w$, upper-bound) then ▷ Verify the senior roles of the role $w$.
        $b \leftarrow$ VerifyAuth($w$, $AR_w$, lower-bound) ▷ Verify the junior roles of the role $w$.
    end if
    Output $b$      ▷ $true$ if $w$ is administered by the authorised administrator, or $false$
    otherwise.
```

where P_r denotes $\mathsf{ID}_r||\mathcal{PR}_r||\mathcal{SR}_r||AR_r$, ID_r is the identity of the regular role within the range, $\mathcal{PR}_r$ is the set of the identities of r's immediate senior roles, $\mathcal{SR}_r$ is the set of the identities of r's immediate junior roles, AR_r is the identity of the administrative role who can administer the range, and $\mathcal{D}_a(P_r)$ is the the parameters' signature which is issued by the administrative role AR_r. These parameters are computed and attached to the role when this role is created or modified by the administrative role.

Assume that a user of the system wants to encrypt some data to a role, or decrypt some data that are encrypted to a role. Then the user will need to verify the parameters AP of the role to check whether this role is certified by the authorised administrative roles of the system. Now we describe an algorithm, Algorithm 1, for this verification process. In Algorithm 1, we define a function to denote the verification of an ID-based signature. This function $verify(\mathcal{D}_z, \mathsf{ID}_z)$ verifies a signature $\mathcal{D}_z$ against the identity of the role z; it returns $true$ if the verification succeeds, and otherwise returns $false$. When running this algorithm to verify a role, a user gets $true$ if the last change of the role was made by authorised administrative roles, and otherwise gets $false$. In addition, creating a role out of the specified role range by an administrative role will cause failures in the verification of all the roles which have inheritance relationships with this role.

In RAM, the parameters AP associated with each role specifies only one administrative role. In the example shown in Table 6.3, the authority range of the role DSO is the superset of that of the role $PSO1$. We require that the private key of the role $PSO1$ is known to both the role DSO and $PSO1$. When DSO modifies the roles in $(ENG1, PL1)$, it will need to use the private key of $PSO1$ to sign the parameters AP associated with the roles. Similarly, DSO needs to have the access to the private key of the role $PSO2$. Therefore, we require that the identity of the administrative role specified in the role parameters AP is set to the role who administers the smallest authority range that contains the role. Alternatively, a hierarchical ID-based signature (HIBS) scheme [59] can be used to simplify the key management. When the administrative role DSO modifies the roles in $(ENG1, PL1)$, it signs the parameters AP associated with the roles using its own private key, and HIBS schemes allows the signature to be verified against the identity of the role $PSO1$. The signature generated by DSO can also be verified against the identity of the role $PSO2$.

6.4 Conclusion

In this chapter we have proposed a new RBE scheme and a cryptographic administrative model AdC-RBAC to manage and enforce role-based access policies for the proposed RBE scheme. The propose RBE scheme can be used to enforce access policies on encrypted data, which can therefore be used to protect data privacy in a cloud storage system. An important feature of the new RBE scheme is that it allows the administrative tasks to be decentralised, and this feature makes the scheme suitable to be used with our proposed administrative model AdC-RBAC.

Different from existing administrative RBAC models, the AdC-RBAC model can be used in an untrusted environment, and its security is guaranteed by using cryptographic RBAC techniques. AdC-RBAC uses cryptographic techniques to ensure that the administrative tasks such as user, permission, and role management are performed only by authorised administrative roles; that is, only the administrators who have permissions to manage a role can add or revoke users to or from the role and owners can verify that a role is created by qualified administrators before giving out their data. Other parties who do not have permissions cannot perform administrative tasks to modify RBAC systems and their policies.

7

Applications of Role-based Encryption

With RBAC models, access policies can be specified by different entities in a system depending on the security requirements. Generally, there are two different types of systems depending on which party defines the access policies. Typically in organisation-based systems, administrators specify the access policies that govern which users can upload and view the data stored in the cloud. We will refer to such systems as admin-centric systems. Then there are systems which are open to public users for the purpose of data storage; with these systems, users are allowed to specify their own access policies on the data they want to store in the cloud. Since the access is controlled by users in this type of systems, we refer to these systems as user-centric systems. This chapter considers application scenarios from both these types of systems to illustrate how our proposed RBE schemes can be used to secure the data stored in the cloud. First we briefly describe an admin-centric application scenario involving a banking organisation.

Then we describe in detail a user-centric application scenario involving secure storage of patient-centric health records in the cloud.

This chapter is organised as follows. In section 7.1, we describe an admin-centric cloud-based document sharing system in a banking organisation based on an European bank. In section 7.2, we consider the user-centric healthcare scenario, and present an overview of an electronic patient record system called the Personally Controlled Electronic Health Record (PCEHR) System recently developed by the Australia Government. In Section 7.3, we give the design of our secure cloud-based personal health record (PHR) data storage system, and show how the issues of the PCEHR system can be addressed by our proposed system. Section 7.4 concludes the chapter.

7.1 RBE-based Cloud Storage System for a Banking Organisation

In this section, we describe a cloud storage system that uses RBE for secure document sharing in a bank. The roles and the hierarchy are based on the case studies in [121, 80], which describe a RBAC policy structure for a branch in an European bank. We show how our RBE schemes can enforce the RBAC policies in a cloud storage system for such a banking application.

7.1.1 System Structure

The list of roles defined in the system is shown in table 7.1, which only lists part of the roles in a branch of the bank. Each branch has been divided into several business divisions, and we only consider two divisions, *Financial Analyst* (FA) and *Office Banking* (OB), in this example. Since the roles are the same in each division, the example can be easily extended to applications with more divisions. Each business division has eight roles which are defined as follows:

- Two managerial roles *Head of Division* and *Group Manager*. For example, in the FA division, the two managerial roles are *FA-HOD* and *FA-GM*. They manage

Function	Position	Role
Financial Analyst	Head of Division	FA-HOD
Financial Analyst	Group Manager	FA-GM
Financial Analyst	Specialist	FA-Special
Financial Analyst	Assistant	FA-Asst
Financial Analyst	Senior	FA-Senior
Financial Analyst	Junior	FA-Junior
Financial Analyst	Clerk	FA-Clerk
Office Banking	Head of Division	OB-HOD
Office Banking	Group Manager	OB-GM
Office Banking	Specialist	OB-Special
Office Banking	Assistant	OB-Asst
Office Banking	Senior	OB-Senior
Office Banking	Junior	OB-Junior
Office Banking	Clerk	OB-Clerk
Financial Analyst	-	FA
Office Banking	-	OB
Branch Employee	-	Employee

Table 7.1: List of Roles for a Bank Branch

the resources for the division.

- Five non-managerial roles. For example, in the FA division, the five non-managerial roles are created for different positions in the division as *FA-Asst*, *FA-Clerk*, *FA-Junior*, *FA-Senior*, and *FA-Specialist*.

- A role for the business division. This role is inherited by all the other roles in the same business division. For example, all the roles in the FA business division inherit from the *FA* role.

In each branch, there is a separation of privilege constraint that a user may not be assigned to more than three non-managerial roles. Apart from these eight roles for each division, each branch has a role called *Employee*, and all other branch-specific roles are inherited from the *Employee* role. The role hierarchy of the bank branch is shown as in Figure 7.1. The direction of the arrows indicates the inheritance relationships among these roles in the system. For example, the role *FA-HOD* inherits from the role

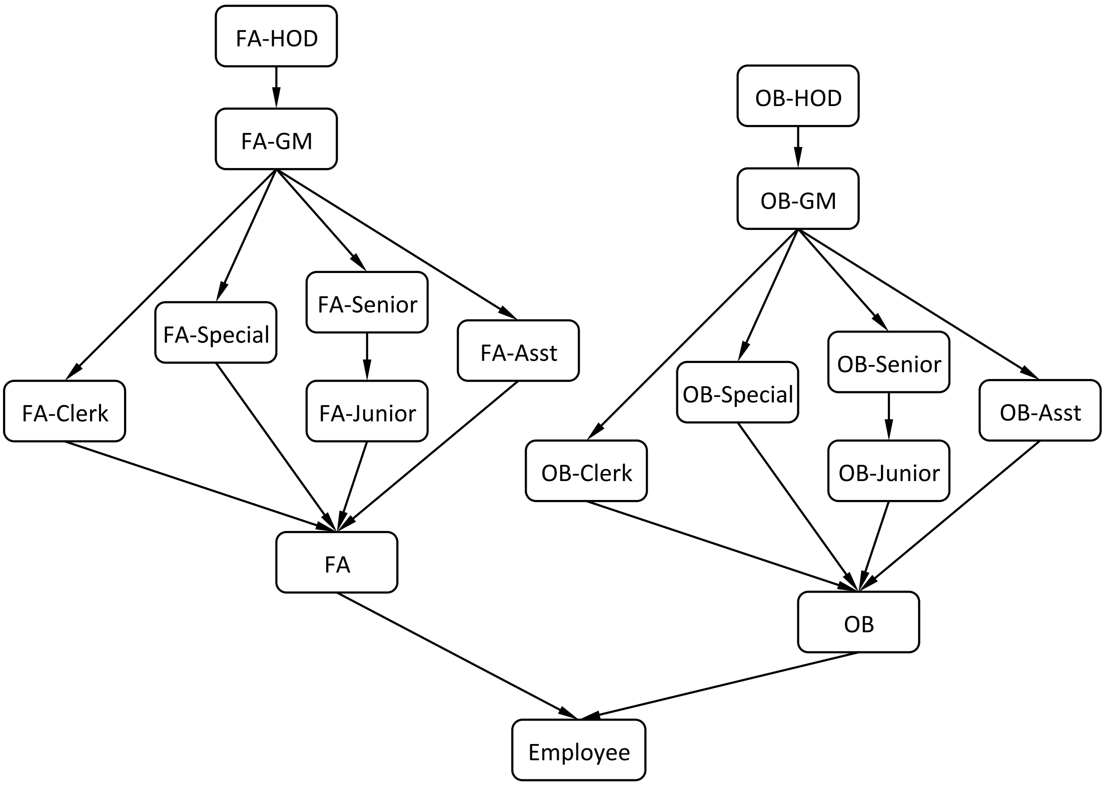


Figure 7.1: A Hierarchical Access Structure for a Bank Branch

FA-GM.

In the branch of the bank, the privilege of each business division is separated from other divisions; that is, a role within a division should not be able to access resources which are assigned to roles within other divisions. In each division, only managerial roles can publish documents in the cloud storage system. Other non-managerial roles, including the role for the division and the role *Employee*, are only allowed to read the published documents which they have the permissions to access.

7.1.2 System Operations

In this subsection, we discuss how to use our RBE scheme to secure the cloud storage system. We discuss several cases where the components of the system and their relationships change dynamically. First, we assume that the administrators of the bank have created the system parameters and the role hierarchy for the branch, and generated the role secret for each role in the branch.

In our RBE schemes, the data owners can encrypt data to any role by using the public role parameters as the encryption keys. These public role parameters comprise different parts for different uses. Among them, one part of the parameters is used for encryption, and one part is used for decryption. In this system, since only the managerial roles can store documents into the cloud, we ensure that the parts for encryption in the role parameters are only accessible to the managerial roles of the division. To achieve this in the framework of using RBE schemes, we assume that the administrators encrypt the role parameters for encryption to the managerial roles of each division while setting up the system. The administrators only need to re-encrypt these role parameters when the role hierarchy changes in a division.

Next we discuss several cases showing how the branch can utilise the system for document sharing.

Case 1 *The division FA publishes a document for the clerks of the division.*

When the division FA has a new policy document for the clerks of the division to view, the document will be given to a group manager first. Since we have assumed that all the role parameters for encryption have been encrypted to the role *FA-GM* using the RBE scheme when the system was set up by the administrators, the group manager can use her or his own decryption key to recover the role parameters for the role *FA-Clerk*. Then the group manager encrypts the document to the role *FA-Clerk* using the RBE scheme and uploads it to the cloud. When a clerk wants to view the document, she or he simply uses her or his decryption key to decrypt the document and view. Any other roles in the branch, except the role *FA-HOD* and *FA-GM*, will not be able to decrypt the document.

Case 2 *The division FA publishes a document for all the employees in the branch.*

When the division FA wants to share a document with other divisions of the branch, the group manager who has been given the document decrypts the role parameters of the role *Employee*, and encrypts the document to the role *Employee*. Recall that our

RBE schemes support role inheritance; that is, the document encrypted to the role *Employee* can be decrypted by any role that inherits from *Employee*. Assume that a group manager of the division OB wants to view the document, she or he can use her or his decryption key to decrypt the document directly.

Case 3 *A new head of division has joined the FA division.*

Now let us assume that the branch has newly assigned a head to the FA division. This new head will be given a decryption key corresponding to her identity by the administrator when she joins the system, and her identity will be included in the updated role parameters for the role *FA-HOD*. The new head can now use her own decryption key to view the documents for the division including the ones which were uploaded before she joined the system. Note that only the role *FA-HOD* needs to update the role parameters, and other roles are not required to take any action.

Case 4 *An assistant in the FA division has resigned from the bank.*

Let us assume that one of the assistants in the FA division has resigned from the bank. The role parameters for *FA-Asst* need to be updated to exclude the identity of this assistant. Even though this assistant still holds the previous decryption key which was able to decrypt the documents encrypted to the role *FA-Asst*, she will not be able to use it to decrypt any future documents after the role parameters are updated. Similarly, only the role *FA-Asst* needs to update its role parameters and none of the other roles in the branch needs to make any change. In addition, other assistants in the same division do not need to update their keys, and hence are not affected by the leaving of this assistant.

Case 5 *A specialist has been assigned to both the FA and OB divisions.*

Assume that the branch has recruited a specialist who will be working for both the FA and OB divisions. First she will be given a decryption key which is associated to her identity by the administrator. Then her identity will be included in the updated role parameters for both the role *FA-Specialist* and *OB-Specialist*. Note that the specialist only needs to keep a single key for decrypting documents which are encrypted to any of these two roles and their descendent roles. Similarly, if the specialist has been assigned to other business divisions of the branch, her decryption key does not need to be changed, and the key size remains constant regardless the number of roles that she belongs to at the same time.

Case 6 *A group manager in the OB division has been appointed as a group manager of the FA division.*

Now consider the situation where a group manager of the OB division has been assigned to manage the division FA. Two actions need to be performed for this position change. The role *OB-GM* needs to revoke this user's role membership, and the role *FA-GM* will need to grant the role membership to this user. To account for this change, both the roles *OB-GM* and *FA-GM* need to update their role parameters, but the decryption key of this user remains unchanged. This decryption key cannot be used to decrypt any future document for the OB division, but can decrypt documents for the FA division from then on.

7.2 Secure Electronic Health Record System

In recent years, there is an increasing trend in the use of the cloud to store personal health records (PHR) online. A number of cloud providers has started providing such services which allow patients' PHR data to be used more effectively, such as the Microsoft HealthVault [101], Google Health[1] [61] and WebMD [132]. In Australia, the Government has recently announced an electronic patient record system called the Personally Controlled Electronic Health Record System (PCEHR) [62] to assist patients in better organising their PHR and provide the patients with flexibility in controlling the access to their PHR.

In this section, we first present an overview of the PCEHR system by describing the involved components and the approaches used by the system to ensure the security

[1] Google Health was closed in 2012 due to the lack of widespread adoption.

of the system. We review the existing access control mechanisms used by PCEHR, and then describe several challenges and issues of the PCEHR system.

7.2.1 PCEHR System Overview

A typical way to store patients' personal health records (PHR) that is adopted by most of the healthcare providers is to use centralised databases that are running on local storage servers. Since each healthcare provider keeps the PHR individually, it is usually impossible for them to share efficiently the medical records of patients. For example, assume that a patient has done the blood test in one clinic, and later on if she or he goes to see a doctor in a different clinic, it is most likely that she or he will need to do the blood test again because as it stands there is no easy way for these two clinics to share electronically the medical records of this patient. Another issue is that the healthcare providers usually do not allow patients to view their health records directly in their databases. Moreover, patients are not able to know whether their PHR data has been accessed by unauthorised persons. Using the cloud to store PHR could make the data sharing among different healthcare providers and patients easier. However, security is the main concern when it comes to storing the PHR in the cloud.

In order to provide an easy and quick access to health information for individuals, the Australian Government launched a personally-controlled electronic health record (PCEHR) system in July 2012. The aim of this national PCEHR system is to allow patients to share efficiently their health information with doctors, hospitals and other healthcare organisations. The PCEHR system enables secure sharing of health information between a user's healthcare providers, while enabling the user to control who can access their PCEHR data. Figure 7.2 shows an example scenario of using the PCEHR system among several parties. Patients store their health information in the PCEHR system, and the other parties may wish to access this data for various purposes. For instance, when a patient goes to see a general practitioner (GP) or the specialist, she or he may wish that the GP or the specialist access her or his PCEHR data directly from the PCEHR system. In the case of an emergency, there needs to be a emergency access policy which will allow the emergency department to access the

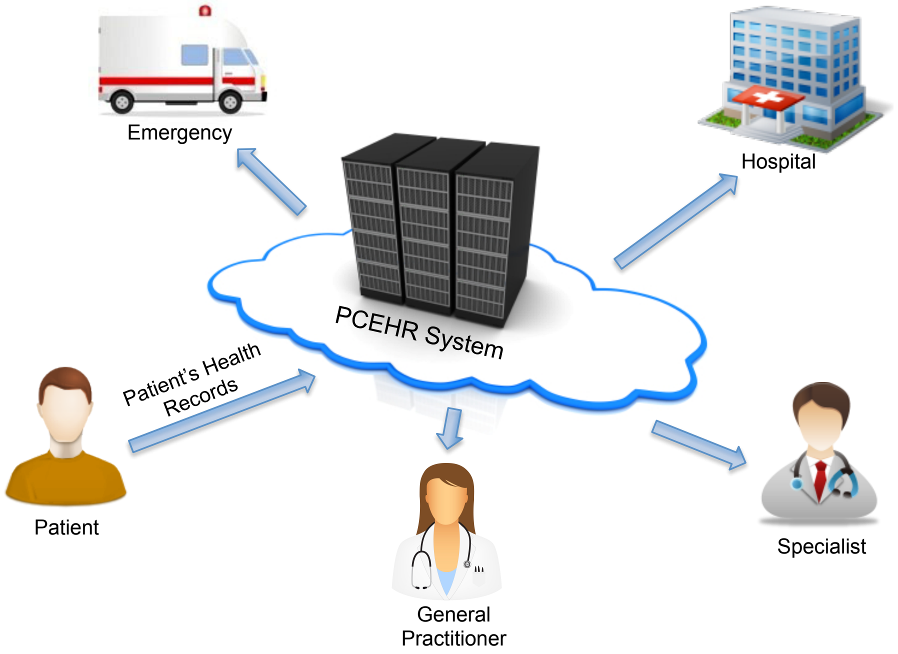


Figure 7.2: PCEHR System Example Scenario

patient's PCEHR data directly. Meanwhile, a hospital may want to access the patient's health information for research purposes.

Let us now review the operations of the PCEHR system to see how these various interaction scenarios are developed in the PCEHR system.

There are two main types of participants in the PCEHR system, namely individual users who wish to store their health information in the system, and healthcare providers and organisations who wish to access the stored health information in order to provide healthcare services to individual users. The users use a Consumer Portal interface to access the PCEHR services. Through the portal, users can view their health information stored in the system, share information with their healthcare providers, manage their access control settings, and view the access history on their PCEHR data. Healthcare providers can access the PCEHR system via either the Provider Portal interface provided by the PCEHR system or the local clinical systems built and owned by themselves. The healthcare providers can seek permissions to access users' PCEHR data, search and view the PCEHR data of individual users, and upload clinical documents.

In the PCEHR system, the PCEHR data is a collection of health documents stored in a network of connected registered repositories. A multi-layer approach that consists of firewalls, gateways, and portals is used to ensure only authorised users can access the PCEHR system. An authorised user who wishes to access the PCEHR data on behalf of a healthcare organisation needs to be authorised by the organisation first. When an authorised user wishes to access the PCEHR system from the clinical system of the organisation, the local clinical system needs to authenticate itself to the PCEHR system using the organisation's digital credentials and pass on the user details. If authorised users want to use the Provider Portal to access the PCEHR system, they need to present a National Authentication Service for Health (NASH) token (e.g. a smart card or USB token) asserting their identity to log in. Individual users/patients will make use of username/password authentication process combined with challenge-response using shared knowledge questions and one-time passwords to log into the PCEHR system.

The protection and security of users' personal information are the responsibility of the system operator who is in charge of establishing and operating the PCEHR system. The system operator is committed to keeping secure the personal information that is stored in the PCEHR system, and is acted by the Secretary of the Department of Health and Ageing. The system operator will take reasonable precautions to protect the personal information it holds from misuse and loss and unauthorised access, modification or disclosure. Customer service officers from the Medicare area of the Department of Human Services (DHS-Medicare) will undertake some of the PCEHR system's daily tasks on behalf of the system operator. In addition, the system operator is authorised to prepare and provide de-identified data for research and other public health purposes. The PCEHR rules will ensure that appropriate protections are put in place around the preparation and disclosure of de-identified data.

7.2.2 Access Control in PCEHR System

The PCEHR system allows the users to choose from two different methods to specify their access control policies for their health records, namely basic access control and advanced access control. Let us now review the access control mechanisms that are currently used by the PCEHR system.

Access List

The core of the access control in the PCEHR system is the “access list”. A user has the flexibility to add or remove organisations to or from the access list. Only the organisations on the access list are permitted to access the user’s PCEHR data. Users can see their access list and update it at any time via different channels. The organisations on the access list will be removed automatically if they have not accessed any of the user’s PCEHR data for three years or more, and the user needs to grant the permission to the organisations again if she or he wishes to allow the organisations to access the PCEHR data. Note that the access control policy is enforced by the system itself in the PCEHR system.

Basic Access Control

When users choose to use basic access controls, the PCEHR system operates on a “care-based access” model. In this model, users do not need to manage the access list explicitly. Any healthcare organisation involved in the care of a user is added to the user’s access list automatically. To make users aware of what happens to their PCEHR data stored in the system, the PCEHR system gives the users the option of setting up a range of notifications when: 1) a new organisation is added to the access list; 2) new PCEHR data has been uploaded to the system; 3) their PCEHR data has been accessed by a nominated representative.

Users can request the healthcare providers not to upload clinical documents to the PCEHR system if they do not want to share the documents. Users should also be informed when a healthcare provider finds that a clinical document may be inappropriate to be added to the users’ PCEHR. If a clinical document has already been added to a user’s PCEHR, and the user wants to remove it from the system, the user can request the PCEHR system operator to remove the document. If the request comes with a legal reason, the system operator will lock the document to prevent future access by any user and healthcare organisation. The “removed” clinical document can be later restored by the system operator upon the request of the user.

Advanced Access Control

Advanced access control settings are the combinations of basic access control settings and some additional access controls.

In basic access control, healthcare organisations are added to a user's access list automatically if they are involved in the care of the user. In advanced access control, the user is able to mark an organisation on the access list as being "revoked" if the user does not want the organisation to access her or his PCEHR data. In addition, users are able to specify if they want their PCEHR to be "findable" or not. If a user set her or his PCEHR to be "not findable", any future healthcare organisation which is not currently on the access list and any "revoked" organisation will not be able to find the user's PCEHR data, and any search for the user's PCEHR will return nothing. By default, a PCEHR will be set to findable, and users can change the setting after the PCEHR data is loaded into the system.

Provider Access Consent Code (PACC): Under the advanced access control settings, users are able to set up PACC which is used as a PIN or passphrase. Organisations will not be able to add themselves to the access list unless they have the valid PACC. The organisations which have already been added on the access list can access the users' PCEHR with valid PACC even if they are marked as "revoked" on the list. The following question needs to be answered if a user chooses to set up a PACC,

- *If you forget your PACC, do you wish participating organisations to be able to access your PCEHR by obtaining your consent?*

If the user responds "no", organisations will not be able to access the user's PCEHR unless the valid PACC is presented. If the user chooses "yes" to the above question, the participating organisations will still be able to access the user's PCEHR without the valid PACC, but only limited access will be granted which we will discuss below. Users can choose the setting about whether to be notified if an organisation has accessed their PCEHR data without their PACC.

Document Level Access Controls: The PCEHR data of a user will have different levels of access controls if the user chooses to set up a PACC. The access control level is set on each clinical document for each healthcare organisation. The available levels are as follows:

General access: The clinical documents of this level are accessible by any healthcare organisation which has access to the user's PCEHR. For example, when an organisation accesses a user's PCEHR data without the valid PACC, only the clinical documents with *general access* can be accessed by the organisation.

Limited access: The clinical documents of this level are accessible only to a more limited group of healthcare organisations as specified by users. The healthcare organisation that uploaded the clinical documents can access the documents regardless of the access control level of the documents. Clinical documents can be set as *limited access* to nominated representatives in the healthcare organisation so as to ensure only the authorised representatives can access the documents.

The access level to clinical documents is managed by users. Healthcare providers do not need to specify the access level while uploading documents. The access level for a clinical document is set when it is uploaded to the PCEHR system, and the default access level is set to be the same as the level of access that the healthcare organisation has when it uploads the document. In some cases a user may want to set different access levels to the documents from the same organisation. In that case, the user can change the access level accordingly after the documents have been uploaded to the system.

By default healthcare organisations on the access list have access to the users' clinical documents with a "general access" level. In order to access "limited access" documents of a user, a healthcare organisation needs to obtain a special provider access consent code (PACCX) from the user at the point of care. Users can reset their PACCX if they forget the PACCX or they want to choose a new PACCX. Creation of a PACCX is only available to users who have opted to set up a PACC.

Emergency Access

It is important to ensure that healthcare provider can access user's PCEHR data regardless of the access control settings in the case of an emergency where the user is not capable of giving or communicating consent.

Emergency access will override the access control policies that are set on users' PCEHR data. However, it is not required for the users that are using basic level access controls, as the organisation who are providing healthcare services to the users will be granted the access automatically. It is required for the users who have set up advanced access controls which may prevent access to their PCEHR data in an emergency situation. With the emergency access, those organisations which are not on the access list or have been marked as "revoked" as well as the ones without the PACC can have unlimited access to all the users' PCEHR data except the documents that have been "removed" upon the user's request previously.

Emergency access will add the healthcare organisation to the users' access list if it is not already on the list and provide the organisation with access to both "limited access" and "general access" clinical documents. The organisation's access level will be reverted to the previous access level prior to emergency access after a period of five days from the time of last access to the user's PCEHR data. If the access is still required after the timeout, the healthcare organisation could assert emergency access again. There is no limit on how many times the emergency access can be used, though the organisation may want to obtain a more persistent form of access from the user or her or his authorised representatives.

To prevent the abuse of emergency access, all use of emergency access will be audited and recorded by the PCEHR system.

Forward Consent

When a user is referred by a healthcare organisation to another organisation, it may be necessary for the referred organisation to have access to the user's PCEHR before the user's visit. In order to have this access, the referred healthcare provider needs to contact the user and obtain the PACC to add itself to the access list of the user.

7.2.3 Challenges and Issues

In the PCEHR system, users trust the system and the operators to protect the security and privacy of their sensitive health information. However, since PCEHR data is stored in a distributed manner over a network of multiple registered repositories, risks can arise as one repository may be compromised or employees of the distributed system may access the PCEHR data without the users' permissions. In addition, flaws can also arise in system implementations, for instance in the enforcement of access control policies that are set up on PCEHR data by users. A flaw in the system software could potentially leak the users' sensitive health information to unauthorised healthcare organisations as the data is stored in the plain form in the PCEHR system.

Now let us use the example shown in Figure 7.2 to discuss other security issues that may occur in the PCEHR system. We assume that users are using advanced access controls in all of the following cases.

Limited Access: In order to provide users the capability to have flexible control on their PCEHR data, the PCEHR system allows users to set up advanced access controls where certain clinical documents can be set a limited access level, so that only the organisations with special permits can access. For example, a patient may wish that her or his GP can access only documents with "general access" by giving the general access to the GP, and the limited access to the specialists. Now let us assume that two specialists, a cardiologist and an endocrinologist, are involved in providing care to a user. The user may not want the cardiologist to access the clinical documents uploaded by the endocrinologist. In this case, the user cannot set different "limited access" level to the documents for different specialists due to the limitation of the PCEHR system.

Access Revocation: Assume that a user has moved to a different suburb, and wishes to change to a different GP. In the PCEHR system, a healthcare organisation can still access a user's PCEHR data when it has been "revoked" from the access list by the user if it knows the PACC of the user. The PCEHR system allows users to reset their PACC if they forget the PACC or want to choose a new one. Hence the user can revoke the access from the previous GP by generating a new PACC, and not letting the previous GP know the new PACC. However, this operation will affect all the organisations that have "general access" as the PACC they possess cannot be used to access the user's PCEHR data any more. The user needs to resend them the new PACC to grant them the access to the PCEHR data. In short, users need to distribute the new PACC or PACCX whenever they want to revoke the access of one organisation which knows the PACC or PACCX.

Forward Consent: When a user is referred by a healthcare organisation to another organisation in the PCEHR system, the referred organisation needs to contact the user to obtain the access to the user's PCEHR data. This could be an issue if the referred organisation needs to provide care to a large number of patients. For example, when a large number of users have been referred by their GPs to a specialist, the specialist needs to spend significant efforts in contacting each individual patient for access to their PCEHR data. If a user does not grant the access to the specialist in a timely manner, the specialist may give inadequate or inappropriate care to the user due to the lack of health information of the user.

De-identified Data Sharing: The PCEHR system allows the system operator to reveal users' PCEHR data for research and other public health purposes after removing the identities from the data. To prepare de-identified data has always been a challenge and it may not be safe in all cases. For example, a patient who has had a special disease can be identified easily from her or his health information. Hence some users may wish their PCEHR data not to be revealed to others under any circumstance. Therefore it is necessary for users to decide whether they want to participate in particular research projects.

Now consider a general healthcare information storage system which uses the cloud to store patients' personal health record (PHR). A well-designed PHR storage system should ensure that the PHR data stored in the cloud will be accessed only by the users who are allowed by the access policies. In such a system, when a user wants to view the PHR data of a patient, the patient should be able to grant the user access only to the data that the user needs to view. We list the important security requirements that the system needs to meet in order to provide a secure and flexible PHR storage service as follows:

- The system should allow the patients to have fine-grained access control on their PHR data.
- The system should make the data communication easy among multiple users who can access the data in the system.
- The system should ensure that the stored PHR data can only be accessed by the users who are allowed by the access policies.
- The sensitive parts of the PHR should be stored in an encrypted form and not in plain format.
- The patient should have the ability to easily grant or revoke the access to or from an user.
- The patient should be able to delegate the permission management to other organisations.

7.3 Our Healthcare Information Storage System

To address all the needs discussed above, we have developed a *secure patient-centric PHR storage system* using our role-based encryption (RBE) scheme described previously. This system can provide a practical solution for securely storing PHR data. Our system can work as either a standalone PHR storage system or a security enhancement component which can be integrated with the PCEHR system.

To reduce the reliance on the trust of the system's security, we use cryptographic techniques to encrypt the health information before storing it into the system which could further enhance the security of the PHR storage system. Similar to the PCEHR

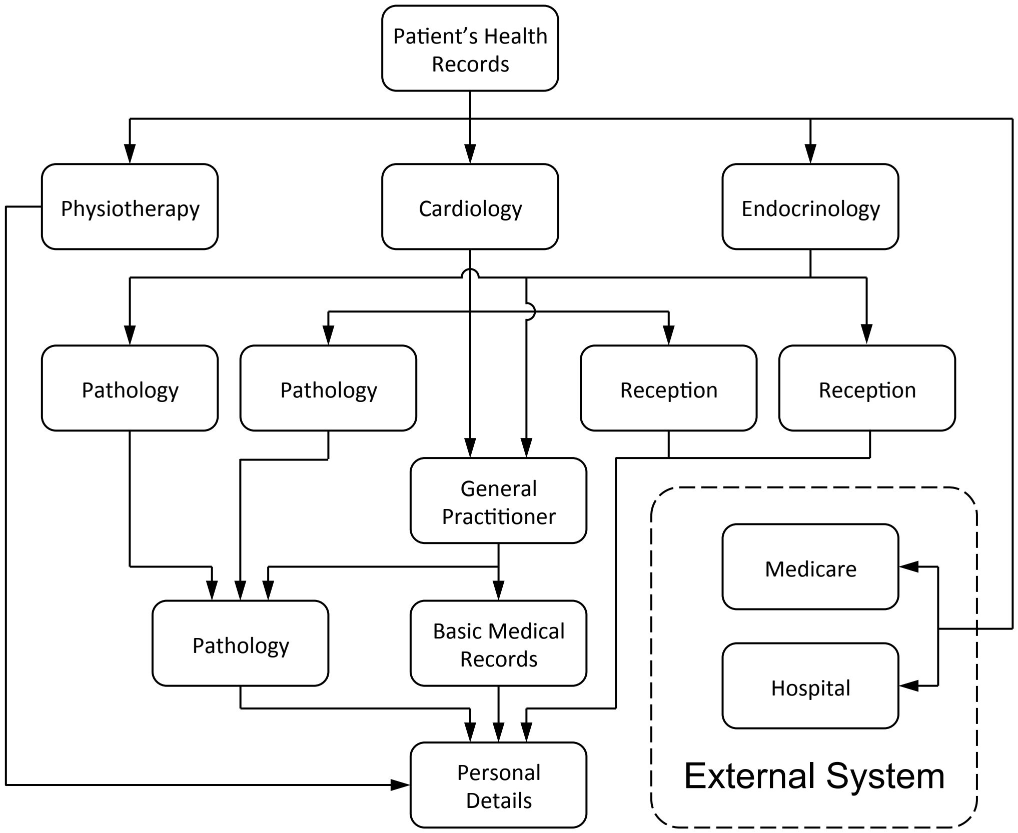


Figure 7.3: PHR Access Structure for the PHR Storage System

system, the PHRs in the system are patient-centric (user-centric); that is, the patients themselves specify who can access their PHR data stored in the system.

We first describe the design of the PHR structure. Then we explain how it satisfies the various security requirements by describing several operations of the system.

7.3.1 Patient-centric PHR Structure

In our system, the PHR data of each patient is divided into several different categories which are organised in a hierarchical structure. Using this approach, patients can define multiple access levels on their PHR data. Let us now consider the design of the patient-centric PHR structure. We let the categories of PHR data be organised in a hierarchical structure shown as in Figure 7.3.

In the designed PHR structure, *Personal Details* stores the personal information of the patient, such as name, date of birth and address. *Basic Medical Records* stores the commonly used records, such as immunisation history, medications and allergies. *General Practitioner* stores the medical records that the general practitioner (GP) of the patient can view. *Cardiology*, *Endocrinology*, and *Physiotherapy* stores the medical records for the specialists to read. *Reception* stores the appointment information and the visit history with the specialists. *Pathology* stores the pathology test requests and result reports. *Medicare* and *Hospital* store the Medicare-related information and PHR data that the patient agrees to share with the hospital for research and public health purposes respectively. *Patient's Health Records* is a category that the patient uses to access all of her or his PHR data.

As specified by the PCEHR operation specification [70], the health information stored in the PCEHR system can be one of the following clinical document types. Now we show how these document types supported by the PCEHR system can be mapped to the categories in the designed PHR structure.

Shared Health Summaries is a clinical document sourced from the user's nominated provider. It provides a clinically reviewed summary of a user's healthcare status as well as information about a user's allergies and adverse reactions, medicines, medical history, and immunisations. The nominated provider is the healthcare provider chosen by the user to provide ongoing care to the user. For example, for the majority of Australians, the nominated provider will be the user's regular general practitioner (GP).

This type of document contains only the basic health information of a user. Hence it should be accessible to most of the parties in the system. As the information is provided by a GP in most of the cases, we let this type of document be stored in the category *General Practitioner*; that is, these documents are encrypted to the role *General Practitioner*, the user's regular GP.

Event Summaries is used to capture key health information about significant healthcare events that are relevant to the ongoing care of a user. Any participating healthcare provider can submit Event Summaries to the PCEHR System.

An event summary is intended to be the "default" clinical document type and is used when none of the other types of clinical document are appropriate.

As this document type is "generic", it can be stored in any category. In general, we let it be encrypted to the role to which the creator belongs. For example, the documents created by a cardiologist will be encrypted to the role *Cardiology.*

Discharge Summaries is the summary information in regards to the discharging of the user from an acute setting hospital. This type of document should be encrypted to the specialist role who is in charge of the treatment of the user so that the specialist will see the summary information of the treatment provided by the hospital.

Specialist Letters are the messages that specialists wish to send directly to the intended recipients. These documents are created by a specialist, and can be encrypted by a specialist to any intended recipient role that users have created.

Referrals are the messages from a healthcare provider to another organisation for the purpose of referring the user to the receiving organisation. Usually, the referrals are created by a GP to refer a user to a specialist. Therefore, in most cases, the documents are encrypted to specialist roles by a GP.

Prescribing and Dispensing Information is a copy of prescription and dispensing information uploaded to the PCEHR system by participating prescribers and dispensers who have access to the system. This type of document will be encrypted to the role to which the doctor who prescribed the medicines belongs. For example, the prescription of Amoxicillin prescribed by a GP will be encrypted to the *General Practitioner* role.

Pathology Result Reports is a copy of the result reports uploaded to the PCEHR system by participating pathologists. We suggest a *Pathology* role for each doctor role, including all the specialists and the GP. The reason is that a patient may not want a cardiology specialist to see her or his endocrinology test result report. However, the *Pathology* role for the specialist roles can inherit from the *Pathology* role for the *General Practitioner* role because the patients may allow the specialist to access the result reports of some basic pathology tests, such as the report for a routine blood test.

Medicare Information is the information provided by the Department of Human Services, such as the Medicare claims history, Pharmaceutical Benefits Scheme (PBS) data, Organ Donor information and the Australian Childhood Immunisation Register. We have defined a *Medicare* role to store all the Medicare related information. This role resides in the external system where the user management permissions are delegated to the managers of individual roles; that is, who can access users' Medicare information is determined by the managers of the Department of Human Services instead of the users themselves.

Consumer Entered Health Summary is the summary information that users wish to share with their healthcare providers, such as their contact details, allergies, and medications. These documents can be encrypted to either the role *Basic Medical Records* or the role *Personal Details* depending on their detail types. For example, the contact details of users are encrypted to the role *Personal Details*, which, for example, the receptionists of doctors can read, and the allergies and medications are encrypted to the role *Basic Medical Records* which only doctors can access.

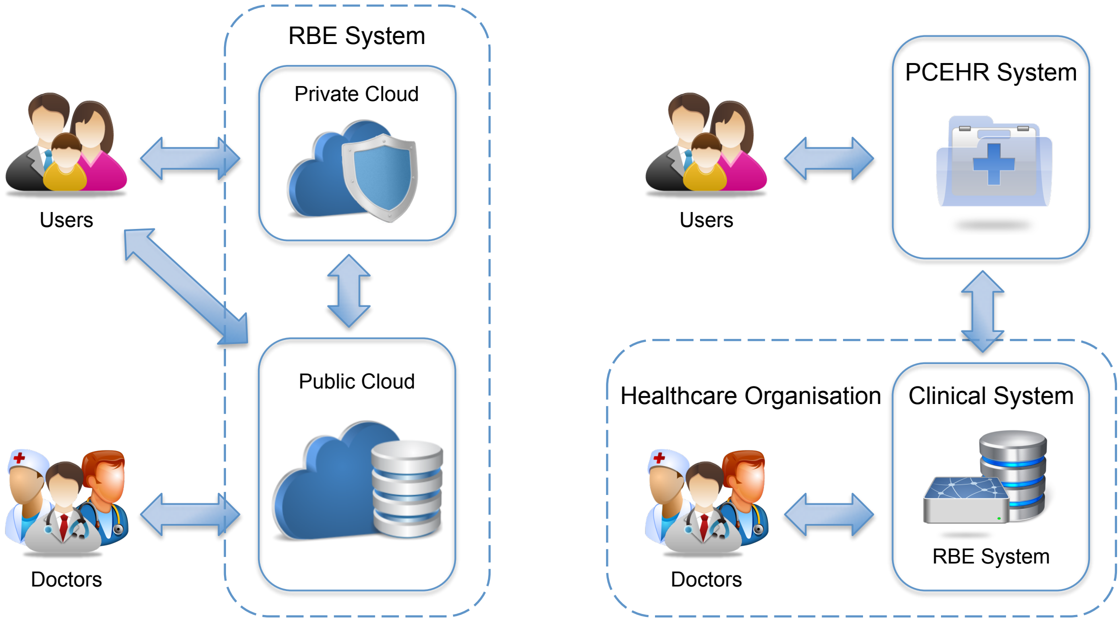
Consumer Notes are the records entered by users as a memory aid for individuals and their representatives, and are not visible to healthcare providers. We let this information to be encrypted to the role *Patient's Health Records* which only users themselves can access.

This hierarchy only shows an example of categories that may be used in practice. The actual categories can be defined by patients themselves depending on their own cases. For example, the list of categories for the specialists can be extended according to the ones that an individual patient wants to see, such as Dermatology and Gastroenterology. Moreover, each specialist category can be extended to multiple levels depending on the certificate level of the specialist. The list of the external organisations can also be extended depending on those with whom the patient wishes to share the PHR data. An external organisation may have its own hierarchical structure. Therefore, when a patient shares PHR data with an external organisation, it will be the external organisation that decides which individual user can access the data. In this example, we only consider such an organisation as a single category for the sake of simplicity, and we assume that the patient does not specify the access policies for the external system, but the patient has the ability to revoke the access from malicious users.

Figure 7.3 also shows the permission inheritance relationships in the system. In general, the users who have been granted access to a category should be able to both read and write the data from or to this category. Consider, for instance, the read permission inheritance. For example, the receptionist of the cardiologist can only read the personal details of the patient whereas the cardiologist should be able to read the medical records written by the GP as well as the basic medical records of the patient. The write permissions can be assigned independently from the read permission. For example, the patient may not want the receptionist to write data to any other categories, and a specialist should have the write access to the appropriate category such as *Pathology*.

7.3.2 System Architecture

Next we discuss how our PHR data storage system protects the privacy of stored PHR data. First, we look at the architecture of our PHR storage system. As mentioned previously, our system can work as either a standalone PHR storage system or a security enhancement component that can be integrated with the current PCEHR system. Our system is implemented using the RBE schemes described in both Chapter 4 and Chapter 5. The system will use one of these two RBE schemes depending on the architecture chosen in the particular application scenario. The architecture for these two cases are shown in Figure 7.4.



(a) Cloud-based System (b) PCEHR-based System

Figure 7.4: PHR Storage System Architecture

Standalone PHR Storage System

The PHR storage system can be deployed in the cloud for providing services to public users as shown in Figure 7.4(a). In this scenario, we adopt the architecture presented in Chapter 5 which is a hybrid of a private cloud and a public cloud. The private cloud is set up and managed by the organisation which provides the PHR storage services, and the public cloud is a chosen third party cloud provider which can provide reliable storage services.

In our system, the definitions of the categories in PHR structure are considered to be important. If a category definition has been tampered with, it may lead to a failure in the access policy constraint. For example, when a doctor encrypts a PHR record to a pre-defined category of a patient, and if a malicious user, who does not have the access to this category, has replaced the definition of the category with the one that she or he has the access to, then the PHR record will be assigned to the wrong category, and the user can access the PHR record even if she or he is not allowed to.

In order to ensure that the category definitions are genuine and up-to-date, we use the hybrid cloud infrastructure in this system. We use a public cloud to store all the encrypted PHR data, whereas the definitions of the categories that the PHR data belongs to and the access policies are stored in a private cloud to which only authorised registered users can make changes. We assume that the public cloud system is honest-but-curious. That is, the public cloud will faithfully execute the protocol, but it may analyse the protocol and try to reveal the data content that it does not have permission to access. We assume that the private cloud is trusted, and it will verify the identities of users who want to modify the stored information in the private cloud.

In this scenario, each user sets up the hierarchical PHR structure and uploads the public parameters to the private cloud. The master secret keys and role secret keys are kept secret by the user. When a doctor wants to upload a clinical document for a user, she or he downloads the role hierarchy definitions of the user and encrypts the document to the appropriate role agreed by the user. When the user or other doctors wish to view this document, the public cloud will compute and return the auxiliary parameters along with the encrypted document. Only light-weight computation is performed on the client side in order to decrypt the document. Standard access controls are implemented on the cloud side to restrict the access to users' health information which is stored in the cloud. Since the data is encrypted before being stored in the cloud, the privacy will be guaranteed even if unauthorised users have accessed the stored data in the cloud.

PCEHR System's Security Enhancement Component

Our PHR storage system can also work with the PCEHR system as a security enhancement component used by individual healthcare organisations. Figure 7.4(b) shows the architecture when a healthcare organisation uses our PHR storage system to access the PCEHR system. In this scenario, we use the PCEHR system to provide storage services instead of the previously described hybrid cloud infrastructure, as the PCEHR system is a trusted system. Our RBE-based PHR storage system will use the RBE scheme implementation described in Chapter 4 as there is only a single repository to store all the information, and this RBE scheme only needs a public cloud to work with. However with this scheme (as mentioned earlier on Chapter 4), when a user is revoked from the system, all the role parameters need to be updated. This is not a major issue in this application, as the number of roles that each user needs to manage is relatively small. Therefore, using this RBE scheme does not affect the performance of the system in terms of user management.

Note that the PCEHR system does not provide any computing service, so we need to have an extra layer between end users and the PCEHR system to provide computing services. Recall that the PCEHR system allows healthcare organisations to access the system via their own local clinical systems. Our PHR storage system can therefore be deployed in local clinical systems of healthcare organisations that want to adopt this solution. We have developed an application which users can install on their client device to manage and access their PHR data stored in the system. To use the system, users can set up their hierarchical PHR structure, upload the public parameters to the PCEHR system, and keep all the secret keys. When a doctor of a healthcare organisation wants to upload a clinical document, she or he will use the local clinical system to encrypt the document, and then upload it to the PCEHR system. When a doctor wants to view a clinical document in the PCEHR system, she or he can download and decrypt the document again via the local clinical system.

7.3.3 System Operations

Now we describe the operations in our PHR storage system to show how they can remedy the weaknesses in the PCEHR system. We assume that each user that uses the system has a unique identity and a public/private key pair of a public key encryption and signature scheme, and all the data that has been written to the cloud has the signature generated by the creator.

Creating User Account: When a patient wants to use the system, she or he first registers online and runs the *Setup* algorithm to generate the system parameters for herself or himself. Then the user can run the *ManageRole* algorithm using the generated master secret key to create the PHR data hierarchy and upload the role public parameters to the system. A separate public/private key pair of the chosen public key encryption and signature scheme will also be generated for the patient.

Organising Patient's Records: After initialising the system parameters, the patient can start using the hierarchy to store the PHR data. For example, the patient runs the *Encrypt* algorithm to encrypt her or his name, address, and contact details to the *Personal Details* role, and encrypt the allergies and medications to the *Basic Medical Records* role. Later on when the patient wants to see a GP, the GP can access this information directly after being added to the role *General Practitioner*, and the patient does not need to re-encrypt the information to the GP.

Before a Doctor Appointment: When the patient wants to make an appointment with a doctor, she or he generates the user decryption key for the unique identity of the doctor by running the *Extract* algorithm and adds the doctor to the proper role by running the *AddUser* algorithm. Then she or he encrypts the generated user decryption key using the doctor's public key, and sends the encrypted key to the doctor. If appointments with doctors of this role need to be made through receptionists, the patient then creates a *Reception* role, and includes the receptionist who she or he contacts as a member. The patient also creates the user decryption key for the unique identity of the receptionist and sends the key to the receptionist. Then the patient sends the appointment request to the doctor or the receptionist, and the doctor or the receptionist decrypts the decryption key for her or him, and confirms the appointment time with the patient by encrypting the appointment information to the role that the doctor belongs to or the *Reception* role of the doctor role.

After a Doctor Appointment: When a doctor wants to write some notes after seeing the patient, the doctor and the patient can decide the role to which the PHR data will be written. Then the doctor runs the *Encrypt* algorithm to encrypt the PHR data, signs the data using her or his own private key, and then uploads the encrypted data to the cloud along with the signature. When a GP needs to refer the patient to a specialist, the notes from the GP can be encrypted to the particular specialist role which the patient will see. A specialist can also write notes to another specialist role. For example, an endocrinologist can encrypt some notes to the role *Physiotherapy* if the patient needs to be sent to a physiotherapist for treatment.

Sharing PHR Data: An external organisation that wants to access the patient's PHR data needs to send a request to the patient first. If the patient agrees to share the data, she or he runs *CreateRole* algorithm to create a role for the organisation and sends the role secret key to the organisation. All the users in an external organisation need to obtain the decryption keys either from the patient directly or through the organisation. The organisation can also request a shared decryption key, and give it to all the users in the organisation. With the decryption keys generated by the patient, users can run the *Decrypt* algorithm to reveal the patient's PHR data if they are allowed by the administrator of the organisation to access the data.

Emergency Access: In an emergency, most systems are required to provide a direct access to the patient's PHR data regardless of the access policies that the patient has set for the PHR data. To provide such an emergency access, the patient can create a decryption key for a Trust Authority using a pseudo-identity allocated to the authority, and include the pseudo-identity to the *Patient's Health Records* role[2]. When an emergency happens, the emergency department needs to authenticate to a Trust Authority to request the decryption key. When the emergency department finishes using the decryption key, the patient runs the *RevokeUser* algorithm to revoke the access to the *Patient's Health Records* role. Then the patient creates a new key for the Trust Authority with a new pseudo-identity, and includes the new identity to the *Patient's Health Records* role.

Revoking Access: Patients can add users to any role to access their PHR data, and they are also able to exclude users from any role. For example, when the patient changes the GP or the specialist, the previous doctor may not be allowed to access the future patient's PHR data any more. The patient then runs the *RevokeUser* algorithm to exclude the previous doctors from the roles to which they used to belong. If the patient does not want to share the PHR data with an external organisation, such as a hospital, she or he removes the public parameters of any role that is related to the

[2] Different entities can act as the Trust Authority in different architectures. In the standalone mode, a private cloud can act as a Trust Authority to store the emergency access key. In the integration mode with the PCEHR system, the decryption key can be stored in the PCEHR system, and can only be accessed by the emergency access of the PCEHR system

hospital, and the users in the hospital will lose the access to the shared PHR data.

Pathology Test: When a specialist needs a patient to do a pathology test, the particular specialist can search the stored pathology result reports for the specialist role to which she or he belongs as well as the reports for the patient's GP. If the same test has been done recently, the patient may not need to do it again. If the patient does need to do the test, the specialist encrypts the pathology test request to the *Pathology* role for the specialist. Before doing the test, the pathologist can also check if the result of some test can be found in existing result reports. When the test is done, the result report will be encrypted to the same *Pathology* role for the specialist to view.

From the above description, we can see that all the weaknesses discussed previously have been addressed by our system.

- The sensitive parts of a patient record are stored in encrypted form and not in plain format.
- Instead of using the limited access, our system allows fine-grained access controls by using the hierarchical PHR data structure. Patients can set their PHR data to be accessed only by the intended users.
- To revoke the access from any user, the features of the RBE scheme allow none of other existing users to be affected by the revocation of one user. Once the decryption key is generated for a user, it can be used no matter how the access policy changes.
- When a healthcare organisation is referred by another healthcare organisation, the referred organisation only needs to obtain the decryption key from the patient when she or he registers in the organisation for the first time. From then on, the same decryption key can be used to decrypt any future referrals for the patient no matter how the patient changes her or his access control settings.
- The external organisations need to request the access to patients' PHR data in our system, and the de-identified PHR data will be shared only if the patient

agrees to share the data. Even the system operator cannot reveal the PHR data without the permission granted by the patients.

7.3.4 An Example Scenario

In this subsection, we use an example to explain how a patient can use PHR storage service provided by such a healthcare information storage system.

Assume that Alice is a user who wants to use the PHR storage service. Before she uses the service, she needs to register herself and get an account created in the system. Once her account has been created, she will obtain her master secret keys for managing her PHR data from the system, and the system will generate a default hierarchical PHR structure as shown in Figure 7.3 for her on the server side. Assume that she leaves this hierarchical structure unchanged, though she can update the hierarchy using her master secret keys at any time. To grant users access to her PHR data, she can add the users to particular roles in the hierarchy. For example, she first generates a decryption key for her GP's unique identity, and sends the decryption key to the GP via a secret channel. Then she can add her GP to her *General Practitioner* role by updating the role public parameters in the system.

Now assume that Alice is feeling unwell, and goes to see her GP. Her GP gives her the diagnosis, and stores the medical report to Alice's *General Practitioner* role in the system. Assume that the GP wishes to send her to a cardiologist for further checking. Then the GP can write a referral letter to the *Cardiology* role and upload it to the system. Alice then adds the appointed cardiologist to her *Cardiology* role when she has booked an appointment with the cardiologist. When Alice visits the cardiologist, the cardiologist can read the GP's diagnosis from the system. Then the cardiologist gives Alice another diagnosis and writes a new report.

Now we have two cases. In the first case, the cardiologist's report needs to be viewed by Alice's GP. To achieve this, the cardiologist simply encrypts the report to Alice's *General Practitioner* role, then Alice's GP can view the report. Note that other specialists may also be able to see the report, so it is important that the patient agrees if she or he thinks the report only contains general contents. Another case is that the cardiologist report has sensitive information. Alice may allow the cardiologist to encrypt the report to the *Cardiology* role only, in which case only the cardiologist can view the report.

The cardiologist can then issue an invoice to Alice, and write a copy to the *Medicare* role for the claim purpose. The Medicare office will assess the claim by viewing the invoice online directly.

From this example, we can see that Alice does not need to bring any paper form letter or document with her when visiting a doctor. All her healthcare information can be accessed by authorised parties from anywhere at any time. Moreover, she does not need to worry about the leak of her health records, as they are stored in encrypted form in the PHR storage system.

7.4 Conclusion

In this chapter, we have considered two applications of using our proposed RBE schemes in cloud storage systems. Firstly, we described a RBAC system for a banking application example by considering the structure of an existing European bank and presented the role hierarchy in a branch of the bank. We analysed several cases of documents sharing in the branch, and discussed how to use the RBE schemes to protect data privacy in these cases. Then we have considered a secure cloud healthcare data storage system using our proposed RBE scheme. We first reviewed the existing electronic health record services, the PCEHR system, provided by the Australian government, and we discussed a few issues and weaknesses of the PCEHR system. Then we provided our solution for a secure PHR data storage system. We presented a design of the PHR structure which allows the patients to have flexible controls over their stored PHR in the system. We described two different ways in which our proposed system can be deployed; one as a standalone system, and the other as a security enhancement component of the existing PCEHR system of the Australian government. We described the system operations in detail and showed how the weaknesses in the existing PCEHR system can be addressed by our proposed secure RBE system.

8

Owner's Trust Model for Role-based Encryption

Now we consider the design of trust models that can enhance the security of cloud data storage systems which use RBE schemes. In some RBAC systems, roles and their users are managed by administrators who have control over all the resources in the systems. When using our RBE schemes in these systems, all the administration tasks can be centralised and performed by using the master secret keys of the systems. Therefore, if a data owner wants to know if a RBAC system is secure, she or he only needs to determine the trustworthiness of the administrators of the system. However, in large-scale RBAC systems, users and permissions management may be decentralised to individual roles; that is, the administrators only manage the roles and the relationships among them while the individual roles have the flexibility in specifying the user memberships and associated permissions themselves. Hence considering the trust relationships between data owners/users and role managers is important in such large-scale systems. Though a role manager can manage the user memberships and permission assignments for several different roles, we assume that each role manager manages only one role in our discussion for the purpose of simplicity. In general, one can identify four types of trust relationships: data owners' trust in role managers, role managers' trust in users, data users' trust in role managers and role managers' trust in data owners.

In this chapter, we describe the first two trust models: owners' trust in role managers and role managers' trust in users. These two trust models form a natural pair as they consider trust from a data owner's perspective. We refer to these trust models as Owner-Role RBAC and Role-User RBAC trust models respectively. The remaining two models will be described in the next chapter.

The Owner-Role RBAC trust model assists the data owners to evaluate the trust in role managers in a RBAC system and use this trust evaluation to decide whether to store their encrypted data in the cloud for particular roles. The Role-User RBAC trust model helps the role managers to evaluate the trust in users in the RBAC system and use this trust in deciding whether to grant membership to the users. These trust models can not only prevent the owners from interacting with role managers which have a poor track record in terms of carrying out their functions properly, but also assist the role managers to identify the malicious users who caused the negative impacts on the role managers' trustworthiness. This can in turn be used to reduce the risks associated with interacting with the RBAC system for the data owners and help role managers to keep the RBAC system authentic. The Owner-Role RBAC and Role-User RBAC trust models are independent of each other and serve different purposes.

An important feature of the proposed trust models is that they take role inheritance into account. Since our trust models are for cloud storage systems dealing with hierarchical RBAC schemes, the trustworthiness of a role manager is also affected by the historical behaviour of the role managers of its ancestor roles and/or descendent roles. Similarly, the trustworthiness of a user is also affected by her or his historical behaviour in other roles in the RBAC system. Hence in our trust evaluation, we take into account the impact of role hierarchy and inheritance on the trustworthiness of the role managers and users. As far as we are aware, this is the first time such a trust model for RBAC system taking into account role inheritance has been proposed. We also present the architecture of a trust-based cloud storage system which integrates the trust models with a RBE system. Moreover, we describe the relevance of the trust models by considering practical application scenarios and illustrating how the trust evaluations can be used to enhance the quality of decision making by data owners and role managers of cloud storage service.

The chapter is organised as follows. Section 8.1 describes the trust issues in a system that uses RBE schemes and discusses the trust requirements for data owners and role managers. The formal Owner-Role and Role-User RBAC trust models are presented in section 8.2 and section 8.3 respectively. The architecture for our trust enhanced secure cloud storage system is presented in section 8.4. In section 8.5, we illustrate how our trust models can be used in a cloud service application to enhance the quality of security decision making. Section 8.6 concludes the chapter.

8.1 Trust Issues in RBE Systems

By using RBE schemes in cloud storage systems, a data owner can encrypt the data to a role, and only the users who have been granted the membership to the role or the ancestor role of that role can decrypt the data. In this chapter, we assume that the data owners and users reside outside this role system infrastructure (where the roles are being administered). Hence the issues to consider are how the data owners can decide whether or not to trust the role managers in the system and how the role managers can decide whether and how much to trust the users in the system. Owners consider the trust of role managers in order to ensure that their data is secure after being assigned to the roles, and role managers consider the trust of users so that users with negative behaviours are excluded from the roles, which in turn makes owners trust these roles. In this section, we discuss the trust issues that need to be considered by the data owners and role managers of a cryptographic RBAC system.

8.1.1 Data Owners' Trust in Role Managers

In a cloud storage system, owners are the parties who want to share the data. When they encrypt their data to the roles (in an RBAC system), they need to determine the trustworthiness of the role managers to reduce the risks of unauthorised parties accessing their data. For instance, a data owner may choose not to encrypt the data to a specific role if the role manager is found to have "bad" behaviour histories. Let us now consider some of the key requirements that the owner must consider in determining whether a role manager should be trusted or not. From the owner's perspective, a trusted role manager should meet the following requirements.

- *Requirement 1: The role manager should grant membership to users who are qualified for that role.*

  When a data owner encrypts her or his data to a role, the intention of the owner is to allow the data to be decrypted by the users who are qualified to be in that role. Therefore, it is a basic requirement that the qualified users should have the access to the data. The violation of this requirement is detected by checking whether or not the qualified users can decrypt the data. Not granting the membership to a qualified user is therefore considered as a bad behaviour of a role.

- *Requirement 2: The role manager should not grant membership to users who are not qualified to that role.*

  Another requirement that is expected by a data owner is to prevent users who are not qualified from accessing the permissions to decrypt the data stored in the cloud. A trusted role manager should only grant membership to a user when the qualifications of the user are verified. Granting membership to an unqualified user is therefore considered as a bad behaviour.

- *Requirement 3: The qualified users in a role should not leak the data to unqualified users.*

  Even if a role manager grants membership only to the qualified users, it is possible that a qualified user may leak the data to unqualified users. For example, consider the situation whereby a user, who is allowed to access the private information that an owner has stored in the cloud, leaks it to another user to whom the owner does not want to reveal the information. The violation of this requirement is detected if it is found that an unqualified user has knowledge of the data. It may or may not be possible to discover this situation. In general, we assume that it is not possible to track down the user who leaks the data; this implies that all the users in that role will need to be under suspicion when such a data leak is detected.

In a hierarchical RBAC system, a role can inherit permissions from other roles. The users of a role have access to the data encrypted to any of its descendant roles. When a leakage is detected in the data encrypted to one of the descendant roles of a role, the users of this role are also under suspicion as they have the potential ability to cause the leakage. Therefore, when an owner wants to determine the trustworthiness of a role manager, the behaviour histories of role managers of descendant roles of this role need to be taken into account in the evaluation, as the users in this role could be the cause of the leakage of its descendant roles' data which are not reflected in the behaviour history of the role manager of this role.

- *Requirement 4: The role managers of ancestor roles of the role under consideration should be trusted.*

  Since a role's permissions are inherited by all its ancestor roles, when an owner encrypts data to a role, all its ancestor roles also have access to the data. So the data owners need to consider the trustworthiness of not only the role to which they want to encrypt the data, but also of all the ancestor roles of this role, as encrypting data to this role is equivalent to encrypting data to any of the ancestor roles of this role.

8.1.2 Role Managers' Trust in Data Users

Since roles have the role managers to manage their user memberships, it is role managers' responsibility to build up their own reputation. Therefore it is important for each role manager to be able to evaluate the trustworthiness of users. Role managers can exclude malicious users from the roles; so these users would not affect the trustworthiness of the roles. The ability to evaluate the trust of users is also useful when a user wants to join the role. The role manager can determine the trustworthiness of the new user and decide whether or not to grant the membership to that user. The proper management of users can result in a good behaviour history for a role manager, which in turn affects the owners' decisions on the role manager. From the role managers' perspective, a trusted user should meet the following requirements.

- *Requirement 1: The user should not be involved in the event of leaking resources of the role.*

  When a leak of data is detected, we assume that the role manager can track which users have accessed the data but the role manager does not know who leaked the data. Here we say that a user is involved in leaking data if the data was found to be leaked, and this user has accessed the data before the leaking is detected. A user who has been involved in the leaking event m times will be considered as less trusted than the user who has been involved in the leaking n times if $m > n$.

- *Requirement 2: The user should not be considered as untrusted by any role manager in the system, if the user has more than one role.*

  A user may belong to different roles in a RBAC system. Therefore, the role managers of some other roles to which the user belongs may also hold trust opinions on the user. A trusted user is supposed to act consistently in different roles. Though a user may behave well in one role, she or he will still be considered as untrusted if she or he has bad behaviours in the other roles. The trust opinions of the role managers of other roles on a user can support the evaluation of the user's trustworthiness. A role manager who does not have any trust records in regards to a user (e.g. when a new user requests to join the role) will still be able to determine the trust of the user.

8.2 Owner-Role RBAC Trust Model

In this section, we consider the owner trust models for RBAC systems. We define three entities in our models, namely *Owner*, *User* and *Role*. *Owner* is the entity who owns the data and stores it in an encrypted form in the cloud, and *User* is the entity who wishes to access the data from the cloud. Role is the entity that associates users with the access to owners' data, and each role manages the user membership of itself. When we refer to *Role* in such a context we imply role managers. Hence when we say that users are managed by a role, we refer to the managers of the role who determine the user set of the role. Then we present an example to illustrate how the behaviour histories of roles in the RBAC system affect the trust of a particular role that a user wants to interact with.

8.2.1 Trust Model

Now we give the formal definition of the Owner-Role RBAC Trust Model.

Definition 8.1 ***(Interaction)*** *From an owner's perspective, an interaction is a transaction whereby an owner encrypts data to a role, and the role gives the access to the data to qualified users.*

A successful interaction is an interaction where only qualified users in the role to which the data is encrypted or users in the ancestor roles of the role have accessed the data. An unsuccessful interaction is an interaction where an unqualified user has accessed the data. Next we define two types of unsuccessful interactions.

User Management Failure: User management failure is an unsuccessful interaction caused by a role who did not manage the user membership properly; that is, the role did not grant the membership to users even when the users have qualified for the role, or the role manager has granted the membership to unqualified users.

User Behaviour Failure: User behaviour failure is an unsuccessful interaction where the data is leaked to unqualified users. When an owner detects that her or his data has been accessed by unqualified users, she or he may or may not know which qualified user(s) has leaked the data. Here we define *User Behaviour Failure* as such an unsuccessful interaction where the owner does not know which user has leaked the data. If the owner knows who has leaked the data, we consider this unsuccessful interaction as *User Management Failure.*

Definition 8.2 ***(Trust Vector)*** *We define a trust vector to represent the behaviour history of a role as*

$$\boldsymbol{v} = (r, s_M, s_B)$$

In this trust vector, r is the value related to successful interactions that this role has been involved with, s_M is the value related to the *User Management Failure* of the role, and s_B is the value related to *User Behaviour Failure.*

By using the function $\mathcal{E}$ in Equation 2.1, we define the trust function $\mathcal{T}(\boldsymbol{v})$ that represents the trust value derived from the trust vector $\boldsymbol{v}$ as

$$\mathcal{T}(\boldsymbol{v}) = \mathcal{E}(r, s_M + s_B)$$

In order to assist owners to collect feedbacks from other owners, we assume that there exists a central repository in the system to collect the ratings on all the interactions between owners and roles. The feedbacks are stored in the central repository and are available to the owners.

Definition 8.3 ***(Interaction History)*** *We define the interaction history derived from these ratings of a role R as*

$$Hist_{\mathcal{O}}(R) = \{H_1^R, H_2^R, \cdots, H_n^R\}$$

Each entry H_i^R in $Hist_{\mathcal{O}}(R)$ is defined as a pair of parameters $H_i^R = \langle ID_i, \boldsymbol{v}_{i,R} \rangle$, where $\boldsymbol{v}_{i,R} = (r, s_M, s_B)$ is a trust vector that represents the trust record of interactions that the owner ID_i has had with the role R. r is the number of ID_i's positive feedbacks on the interactions with R, s_M is the number of negative feedbacks on the interactions with R due to the *User Management Failure*, and s_B is the number of negative feedbacks on the interactions with R due to the *User Behaviour Failure*.

Assume that an owner ID_i has assigned a resource to the role R. The central repository will increase r in $\boldsymbol{v}_{i,R} = (r, s_M, s_B)$ by 1. However, if the owner later reports a leak of this resource, the central repository will decrease r by 1 first. Then depending on the failure type, the central repository will increase s_M by 1 if the owner knows who has leaked the resource, or increase s_B by 1 if the owner does not know who has leaked the resource.

In section 8.1.1, we have discussed the trust requirements that an owner should consider when deciding whether or not to trust a role. From that discussion, we see that the factors which can affect the owners' decision come from the interaction history of the role with whom owners have interacted as well as its ancestor roles and descendant roles. When an owner evaluates the trust of a role R, the owner needs to consider the following different trust classes.

Individual Trust: Individual trust is a belief that is derived directly from the interaction history of the role R.

When computing the trust of the role R, an owner obtains the interaction history $Hist_{\mathcal{O}}(R)$ of the role R from the central repository. Assume w_o is the weight that the owner ID_k assigns to the feedbacks from other owners. The individual trust value of the role R is computed as

$$T_{\mathcal{O}}(R)^D = \mathcal{T}(\boldsymbol{v}_{k,R}^D), \quad \boldsymbol{v}_{k,R}^D = \boldsymbol{v}_{k,R} + w_o \sum_{i=1, i \neq k}^{n} \boldsymbol{v}_{i,R}$$

The trust vector $\boldsymbol{v}_{k,R}^D$ in this equation is a combination of all the trust vectors in $Hist_{\mathcal{O}}(R)$ with regard to the role R considering the weighting for the ones from other owners.

Inheritance Trust: Inheritance trust is a belief that is derived from the interaction history of other roles that have inheritance relationships with the role.

First we have a look at the inheritance trust where only the interaction history of the descendant roles is considered. When an owner detects a *User Behaviour Failure* with a descendant role R_d of a role R, the feedback that the owner provided should not only be applied to that descendant role R_d, but should also affect the trust of R (as users belonging to the role R also have the access to owner's data assigned to R_d and hence are under suspicion of causing an unsuccessful interaction). Therefore, while evaluating the trust of the role R, the interaction history from all its descendant roles including R_d needs to be considered.

Assume a role R has m immediate descendant roles $\{R_1, \cdots, R_m\}$, and we define a weight vector $\boldsymbol{w}_{R_i} = (w^R_{R_i}, 0, w^R_{R_i})$ where $w^R_{R_i} \in [0,1]$ is the system specified weight between R and R_i. We define the second element of $\boldsymbol{w}_{R_i}$ as zero because the *User Management Failure* is not considered in inheritance trust. We denote the number of users that have been included in the role R_i as n_{R_i}, and the total number of users that have been included in the role R_i and all its ancestor roles as N_{R_i}. Here we assume that the probability that each user violates the trust requirement is the same. The inheritance trust value derived from the descendant roles is computed as

$$T_{\mathcal{O}}(R)^I = \mathcal{T}(\boldsymbol{v}^I_{k,R}), \quad \boldsymbol{v}^I_{k,R} = n_R \sum_{i=1}^{m} [(\frac{\boldsymbol{v}^D_{k,R_i}}{N_{R_i}} + \frac{\boldsymbol{v}^I_{k,R_i}}{n_{R_i}}), \boldsymbol{w}_{R_i}]$$

where $[\boldsymbol{v}, \boldsymbol{w}] := \boldsymbol{v}^T \boldsymbol{w}$ is the usual dot product on $\mathbb{Z}_q^3$.

Combination Trust: To combine these two types of trusts together, we define a combination trust function for a role R as $T_{\mathcal{O}}(R)$. Assume that $w \in [0,1]$ is the weight of the inheritance trust. The trust is first computed as follows:

$$T_{\mathcal{O}}(R)^C = (1-w) \cdot T_{\mathcal{O}}(R)^D + w \cdot T_{\mathcal{O}}(R)^I$$

Consider the scenario where the combination trust of a role is higher than the trust value of one of its ancestor roles. Then the owners will trust this role at the same level

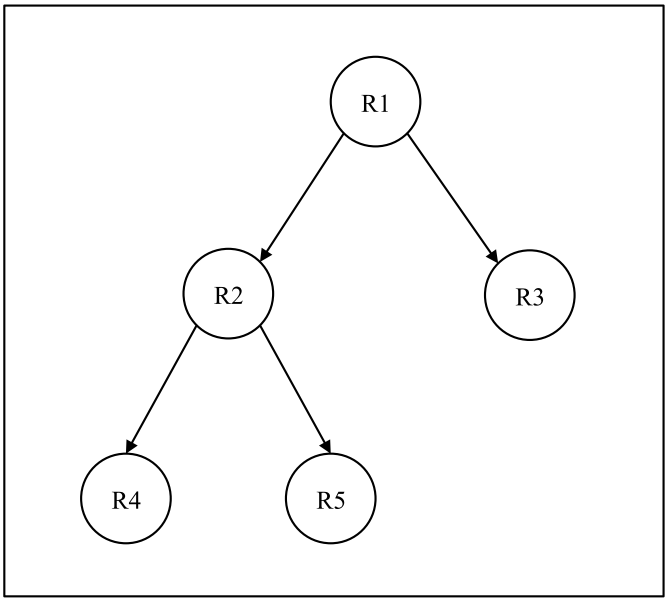


Figure 8.1: Hierarchical RBAC Example IV

as its ancestor role which has a lower trust value, as the users of its ancestor role have the same level of access as the users in this role. So the combination trust of the role will be the minimum value of the trust of this role and the trust of all its ancestor roles. Assume the role R has m immediate ancestor roles $\{R_1, \cdots, R_m\}$. Then the combination trust is amended to be the following:

$$T_{\mathcal{O}}(R) = min(T_{\mathcal{O}}(R)^C, T_{\mathcal{O}}(R_1), \cdots, T_{\mathcal{O}}(R_m))$$

8.2.2 Example

Now we use an example to show how the owners' trust in a role is affected by the feedbacks for different roles in a RBAC system. In this example, we consider all the bad feedbacks as *User Behaviour Failure*, as our intention is to show how the role hierarchy affects the trust value of roles. Consider the role hierarchy example shown in Figure 8.1.

In Figure 8.1, the role R_1 inherits from role R_2 and role R_3, and the role R_2 inherits from R_4 and R_5. For simplicity, let us assume that the number of users in all these roles are the same. We set the weight between every two roles and the weight of other owners' feedbacks to 1; that is, the weight vector for each role is defined as $\boldsymbol{w}_{R_{i+1}}^{R_i} = (1, 0, 1), i \in [1, 5)$, and $w_o = (1, 1, 1)$. When an owner wants to encrypt data to the role R_2, she or he will need to evaluate the trust value of R_2 to decide whether it is safe to give access to the data to R_2. In Figure 8.2, we show the trust values of R_2 when only different individual roles in the RBAC system have feedbacks. For example, the curve for $R_1, GFP = 75\%$ shows the trust values of R_2 when only R_1 in the RBAC system has feedbacks, and 75% of these feedbacks are positive.

When the good feedbacks percentage is 75%, the trust value for R_2 goes up with increasing number of feedbacks. When the feedbacks are only given for R_1, the increase in the trust value is the fastest. This is because all the feedbacks are used in the calculation of the individual trust of R_1, and the combination trust of R_1 is not affected by other roles as there is no feedback for others. Since R_2 has neither individual trust nor inheritance trust, its combination trust is the minimal value among the set which contains the trust value of R_1 only. Therefore, it has the highest value as it is calculated based on all the feedbacks in which the positive ones are in the majority. When the feedbacks are only provided for R_4, the increase in the trust value is the slowest. This is because the feedbacks used in the calculation of the inheritance trust of R_2 has been averaged over three roles, R_1, R_2, and R_4, and only 1/3 of the feedbacks are considered in calculating the trust value of R_2. We see that the trust value of R_2 increases slightly faster when the feedbacks are only provided for R_3. This is because the population in two roles is less than that in three roles, and 1/2 of the feedbacks are considered in the calculation of R_2's trust value. This is analogous to the case where R_2 has more feedbacks where the positive ones are in the majority. When the feedbacks are only provided for R_2, we see that the curve overlaps with the trust value where the feedbacks are only for R_3. Since R_2 and R_3 have the same population in this example, the feedbacks for R_2 and those for R_3 have the same impact on the trust value of R_1. When not considering R_1, R_2 has a higher trust value as the value is based on all the feedbacks. However, taking the lowest trust value among R_2 and all its ancestor roles makes R_2's trust value the same as the one when feedbacks are only for R_3.

When the good feedbacks percentage is 25%, the trust value for R_2 goes down with

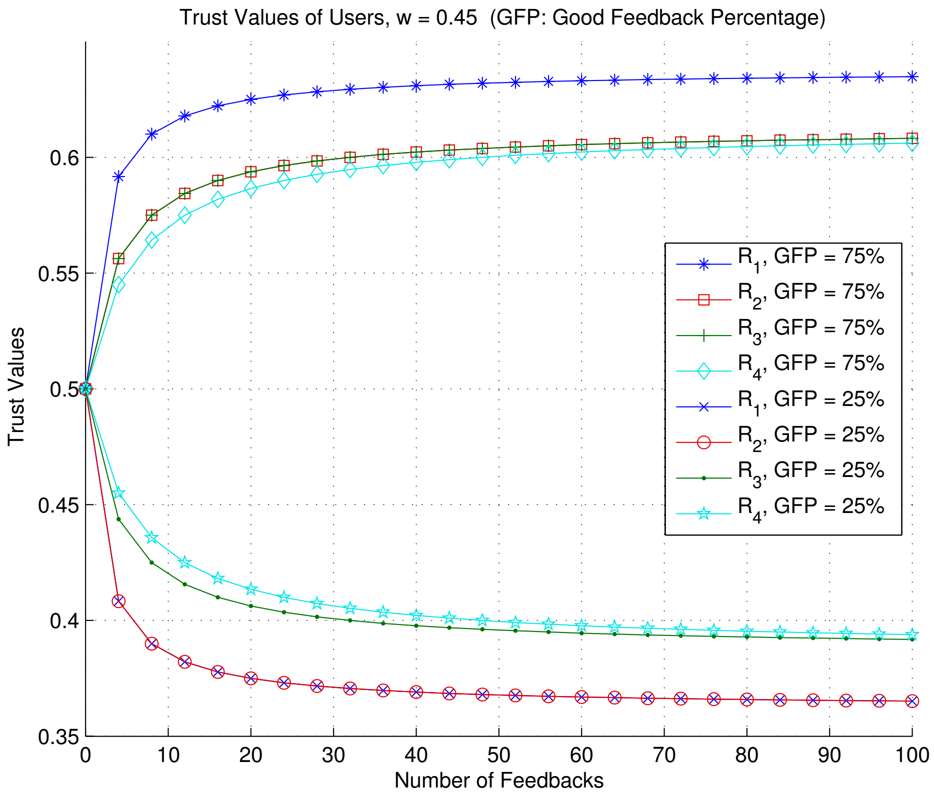


Figure 8.2: Trust Values for R_2 Evaluated by Owners

the increasing number of feedbacks. When the feedbacks were only provided for R_1, the decrease in the trust value is the fastest. Similar to the above, this is because all the feedbacks have been used in the calculation of R_1's trust value which in turn becomes the trust value of R_2, and the majority of these feedbacks are negative. When the feedbacks are only provided for R_4, the decrease in the trust value is the slowest. This is because the feedbacks used in the calculation of the inheritance trust has been averaged by three roles, R_1, R_2, and R_4. We also see that the trust value decreases slightly faster when the feedbacks are only for R_3 because more feedbacks whose majority are negative are used in calculating R_2's trust value. When the feedbacks are only provided for R_2, we see that the curve overlaps with the trust value when feedbacks are only for R_1 instead of R_3. Since the feedbacks for R_2 and those for R_3 have the same impact on the trust value of R_1, the trust value of R_1 is the same as when the feedbacks are for R_3. However, the unamended combination value of R_2 is the same as R_1's trust value (the same as the trust value of R_2) when the feedbacks are only for R_1 as they both are calculated based on the same amount of feedbacks. This trust value is lower than that of the role R_1, and is used as the amended trust value of R_2.

From Figure 8.2, we see that the feedbacks for different roles in the system have different impact on the trust value of R_2. Firstly, the feedbacks on ancestor roles have the most significant impacts on the trust of a role. Secondly, the more users have access to a role's data, the less impact the feedbacks for the role will have on each role that the users belong to. These results show that our owners' trust model is useful in assisting owners to determine properly the trust of roles in RBAC systems.

8.3 Role-User RBAC Trust Model

Since the trustworthiness of a role is primarily determined by the behaviour of users of the role, it is important for the role to ensure that only users with good behaviour are granted the memberships. If roles do not have a way to evaluate the trust of their users, it would be difficult for them to distinguish the malicious users from those with good behaviours. In this section, we present a trust model for roles' trust in users as an extension of the owners' trust model on roles. This trust model aims to assist a role to determine the trust of users who belong to the role or want to join the role.

This trust model can either work independently or work together with the owner's trust model. Roles can use this model to periodically check the trust value of the existing users in the roles, and revoke the memberships from users whose trust values are below the preset threshold. This trust model can also be used by roles to determine the trust value of a new user requesting to join; the request from the users whose trust values are below the threshold will be rejected.

8.3.1 Trust Model

In this subsection, we give the formal definition of the roles' trust in users.

Definition 8.4 ***(Trust Vector)*** *Since there is no interaction between the roles and users, we define a trust vector to represent directly the behaviour history of a user as* *follows:*

$$\boldsymbol{v} = (h, s)$$

In the trust vector, h is the total number of resources that have been assigned to the role, and s is the value related to the leaking events that the user was involved in. Recall that we say that a user is involved in a leaking event of a resource if the resource has been found to be leaked, and this user has accessed the resource before the leaking was detected.

Using the function $\mathcal{E}$ in Equation 2.1, we define the trust function $\mathcal{T}(\boldsymbol{v})$ that represents the trust value derived from the trust vector $\boldsymbol{v}$ as

$$\mathcal{T}(\boldsymbol{v}) = \mathcal{E}(h - s, s)$$

Definition 8.5 ***(Trust Records)*** *We assume that there exists a central repository in the system that collects and stores the behaviour histories of users provided by roles of which the user was a member. We define the trust record provided by a set $\mathcal{R}$ of n roles as*

$$Hist_{\mathcal{R}}(U) = \{H_1^U, H_2^U, \cdots, H_n^U\}$$

Each entry H_i^U in $Hist(U)$ is defined as a pair of parameters, $H_i^U = \langle R_i, v_{i,U}\rangle$ where $\boldsymbol{v}_{i,U} = (h, s)$ is a trust vector that represents the trust record of the user U when she or he is the member of the role R_i. h is the number of resources that have been assigned to R_i when U is the member of the role R_i, and s is the number of leaks related to R_i that U was involved in when U is the member of R_i.

We assume that a role R_k currently has a set $\mathcal{U}$ of n users. When a new user joins the role, R_k will create a trust vector for the user in the central repository. The vector is initialised as $(h, 0)$ where h is the current number of resources that have been assigned to the role.

When a new resource is assigned to R_k, R_k will update the trust records $(H_k^{U_1}, H_k^{U_2}, \cdots, H_k^{U_n})$ in the central repository by increasing h in each vector by 1. When a leak is detected by or reported to the role R_k, R_k tracks the set of users who have accessed the leaked resource, and increases the value s in trust vectors of this set of users by 1.

Since a role can inherit from another role in RBAC systems, besides updating the trust records maintained by R_k, R_k will need to notify all its ancestor roles about the leak, and all the ancestor roles of R_k will update their trust records for their users in the same way that R_k does.

Next we define the trust function used in the model.

Direct Trust: Direct trust of a role on a user U is the belief that is derived directly from trust records of the user U from the role itself.

When a role R_k wishes to evaluate the trust value of a user U, the role first obtains the trust record $Hist_{\mathcal{R}}(U)$ of the user from the central repository. Taking as input the trust record H_k^U maintained by R_k itself, the direct trust value can be computed as follows:

$$T_{\mathcal{R}}(U)^D = \mathcal{T}(\boldsymbol{v}_{k,U})$$

Recommended Trust: Recommended trust is the belief that is derived from the trust records of the user from other roles in the system.

Assume there are n roles $\{R_1, \cdots, R_n\}$ who have provided the trust records for the user. The recommended trust value of the user from the perspective of the role R_k is computed as

$$T_{\mathcal{R}}(U)^R = \mathcal{T}(\boldsymbol{v}_{k,U}^R), \quad \boldsymbol{v}_{k,U}^R = \sum_{i=1, i \neq k}^{n} \boldsymbol{v}_{i,U}$$

Combination Trust: To combine these two types of trust together, we define a combination trust function for a user U as $T_{\mathcal{R}}(U)$. Assume that $w \in [0,1]$ is the weight of the recommended trust. The trust value is computed as

$$T_{\mathcal{R}}(U) = (1-w) \cdot T_{\mathcal{R}}(U)^D + w \cdot T_{\mathcal{R}}(U)^R$$

This trust value is evaluated based on all the trust records in $Hist_{\mathcal{R}}(U)$ considering the weighting for the trust records from other roles.

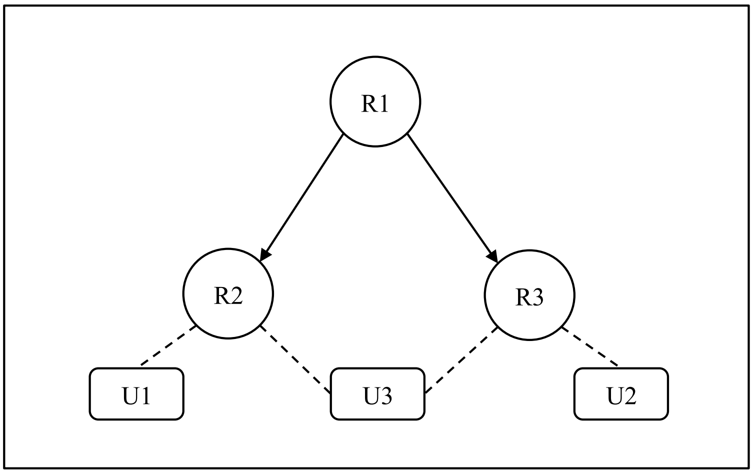


Figure 8.3: Hierarchical RBAC Example V

8.3.2 Example

Now let us consider an example to illustrate how a role's trust in users is related to role(s) that users belong to in a RBAC system. In this example, we assume that we do not know whether these users are the causes of the bad feedbacks. Consider the role hierarchy example shown in Figure 8.3.

In Figure 8.3, the role R_1 inherits from role R_2 and role R_3, the users U_1 and U_2 are the members of the role R_2 and R_3 respectively, and the user U_3 is the member of both the role R_2 and R_3. For simplicity, let us assume that the number of resources assigned to all the roles are the same, and users' feedbacks are from the role(s) to which they belong. For example, U_2's feedback is provided only by R_3 while U_3 is getting feedbacks from both R_2 and R_3. In Figure 8.4, we show the trust values for these users U_1, U_2, and U_3 evaluated from the role R_2's perspective.

When the users' good feedbacks percentage is 75%, the trust values for the users are increasing. The increase in the trust value for U_2 is slowest because the weight of feedbacks from other roles is low where $w = 0.25$. The trust value for U_1 increases faster as the weight for the direct trust in this example is more than that of the recommended trust. The trust value for U_3 increases the fastest as this user is receiving good feedbacks from both roles R_2 and R_3 and the combined value is higher than any one of them. When the users' good feedbacks percentage is 25%, the trust values for the users are

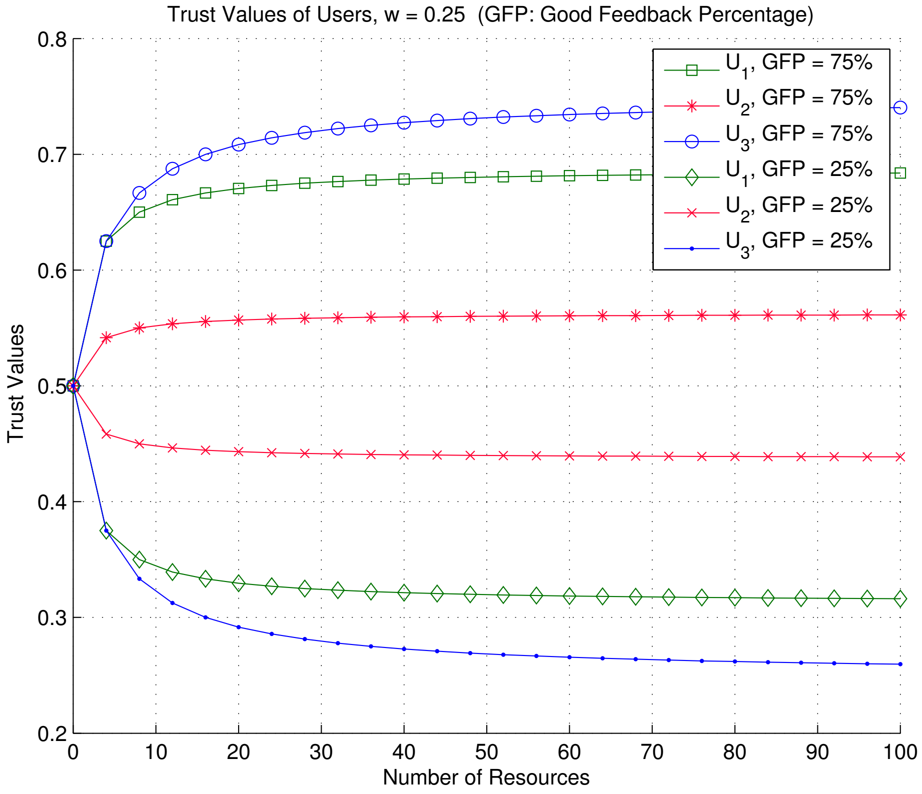


FIGURE 8.4: Trust Values for Users Evaluated by R_2

decreasing. The decrease in the trust value for U_2 is slowest because the weight for feedbacks from other roles is low. The trust value for U_1 decreases faster as the weight for the direct trust in this example is more than that for the recommended trust. The trust value for U_3 decreases the fastest as this user is receiving bad feedbacks from both roles R_2 and R_3.

From Figure 8.4, we see that the feedbacks for users in different roles result in a different trust value when R_2 evaluates the trust of these users. When a user has good trust records in other roles only, the role will trust the user less than another user who has the same trust record in the role itself. A user who has good trust records in both this role and other roles will be trusted the most among the three users. These results show that our roles' trust model is intuitive. Hence it is useful in assisting roles to determine properly the trust of users in RBAC systems.

8.4 Architecture

In this section, we present the design of a secure cloud storage system combining the trust models for RBAC proposed in section 8.2 and 8.3 with a RBE system. This architecture provides a practical solution in building a reliable and trusted RBAC system while retaining the use of cryptographic techniques. We have developed a prototype implementation of this architecture and used it in carrying out our experimental analysis.

8.4.1 System Overview

Consider the system architecture shown in Figure 8.5. Since our trust models are based on RBE schemes, our system contains all the entities that a RBE scheme has, which include an administrator, roles, users, and owners. The administrator is the system administrator of the RBAC system. The administrator generates the system parameters and issues all the necessary credentials. In addition, the administrator manages the role hierarchical structure of the system. To put a role into the role hierarchical structure, the administrator needs to compute the parameters for the role. These parameters represent the position of the role in the role hierarchy. They are stored in the cloud, and are available publicly. Roles are the entities that associate users and owners together. Each role has its own role parameters which defines the user membership. These role parameters are stored in the cloud, and a role needs to update them in the cloud when updating the user membership of the role. Owners are the parties who possess the data and want to store the encrypted data in the 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. Users are the parties who wish to acquire certain data from the cloud. When a user wishes to access stored data in the cloud, she or he first sends

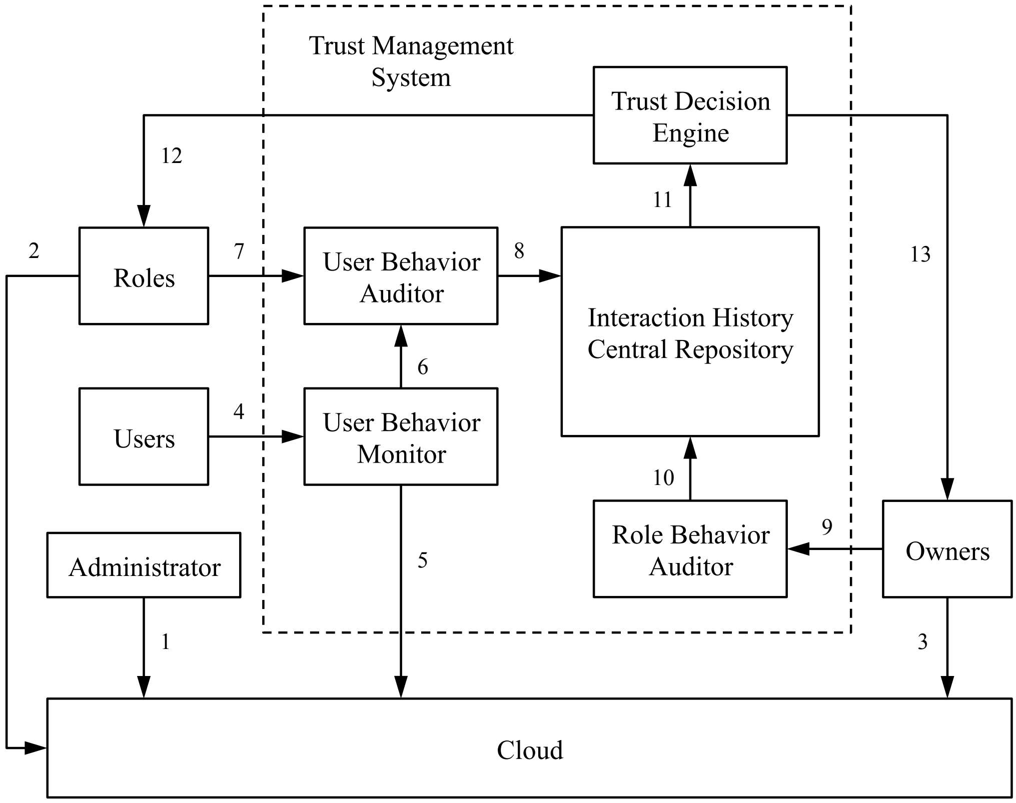


FIGURE 8.5: Architecture for Using Owners' Trust Models in a RBE System

the request to the cloud, and decrypts the data upon receiving the response from the cloud.

In addition to these four entities in a basic RBE scheme, our trust-enhanced RBE system integrates an extra trust management system, which consists of five components. Next, we describe the details of these components.

Central Repository : In our trust models, all the interaction histories and trust records related to roles and users are stored in a central repository. The central repository is used to keep records of all these interaction histories and trust records which are used by the Trust Decision Engine (described as below) in evaluating the trust value of roles and users. Any entity that is residing outside the trust management system is not able to access the central repository.

Role Behaviour Auditor : In order to protect the integrity of the feedbacks on roles, a role behaviour auditor collects the feedbacks for roles from owners. The role behaviour auditor needs to ensure that an owner who uploads feedback is an authorised owner.

All the valid feedbacks will be forwarded to the central repository, and invalid feedbacks will be discarded. Besides the feedbacks from owners, the role behaviour auditor also collects information about data assignment to roles. Owners need to inform the role behaviour auditor when they encrypt data to roles. Then the auditor will update the central repository with the number of resources that have been assigned to the roles.

User Behaviour Auditor : A user behaviour auditor is an entity to collect the feedbacks on users' behaviour. However, unlike the role behaviour auditor, the user behaviour auditor listens on two channels for feedbacks. One is from the roles who may report the leakage of data, and another is from the user behaviour monitor which reports the access histories of users to the stored data in the cloud. This auditor will determine whether a user is involved in the leakage of data, and update the user trust records in the central repository if the user has accessed the leaked data.

User Behaviour Monitor : A user behaviour monitor acts as a proxy server between users and the cloud. It only monitors and forwards the users' requests to access stored data in cloud. When a user wants to access a resource, she or he does not send the request to the cloud directly. Instead, the request is sent to the user behaviour monitor, and the user behaviour monitor will forward the request to the cloud. The monitor will inform the user behaviour auditor the information about which user has accessed which resources.

Trust Decision Engine : The trust decision engine is the entity which evaluates the trust of roles for owners and the trust of users for roles. The trust decision engine takes as input the interaction histories or trust records stored in the central repository, and outputs the trust value of a particular role or user.

8.4.2 System Workflow

All the entities in the system are connected through different communication channels which are labelled with numbers in Figure 8.5. We explain how the system works by describing the information flow through these channels.

First, the administrator initialises the system and specifies the role hierarchy of the system. The generated system parameters are uploaded to the cloud via channel

1. Roles grant the membership to users, and upload role parameters to the cloud via channel 2. Owners encrypt and upload data to the cloud via channel 3. When a user wants to access a resource stored in the cloud, she or he first sends the access request to the user behaviour monitor via channel 4, and the user behaviour monitor forwards the request to the cloud through channel 5. The cloud then communicates with the user as in a normal RBE scheme. The monitor also sends the user behaviour auditor the information about the user identity and the resource identity via channel 6.

When an owner wants to encrypt a resource to a role in the RBAC system, she or he requests for the trust evaluation on the role to the trust management system. Then the trust value of the role will be returned to the owner through the channel 13. If the owner believes that the role is trusted, she or he then encrypts and uploads the resource to the cloud via channel 3. The owner also notifies the role behaviour auditor about the identity of the resource in the cloud and the role to whom the resource is encrypted. The auditor then updates the number of the resources that have been assigned to this role in the central repository via channel 10. When an owner has found a leak in her or his resource to unauthorised users, she or he then provides feedback on the role to whom the resource is encrypted to the role behaviour auditor through channel 9. Once the role behaviour auditor verifies that the feedback is from an authorised owner, it will forward the feedback to the central repository.

When a negative feedback of a role has been raised by an owner because of the leak of a resource, the role will send the identity of the resource to the user behaviour auditor via channel 7. The auditor then updates the trust records of users, who have accessed this resource, in the central repository via channel 8. The role can ask the trust management system about the trust evaluation for a user at any time, and the trust value will be returned by the trust decision engine through channel 12. Upon receiving the trust values for users from the trust decision engine, a role can update the role parameters that represent the user membership in the cloud via channel 2 if there exist malicious users whose role memberships need to be revoked.

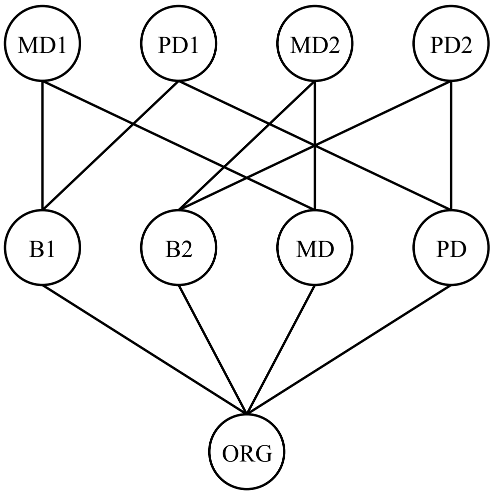
8.5 Application Scenario

Finally, in this section, we consider an application scenario based on a digital library system to illustrate how our proposed trust models can be deployed to enhance the quality of decision making. Assume that the digital library system uses an external cloud storage platform to store all the resources, and publishers are allowed to share their digital resources such as books, magazines, and other types of publications on this platform. A party can subscribe to the publisher for particular resources in order to access the resources stored in the cloud, and the subscription to a publisher needs to be authorised by the publisher. The publisher may reject the subscription request for reasons such as the party is not reliable in paying the subscription or the party has the potential to leak the resources to unauthorised parties.

Now assume that there is an organisation with several branches in different geographical locations and that each branch consists of several departments. When employees of the organisation need to access the digital resources stored in the cloud, the relevant department or the branch (where the employee works) can subscribe to the publisher. Let us assume that the organisation uses a RBAC system to control the access to resources, and the role hierarchy is shown in Figure 8.6.

In this example, the organisation consists of two branches B_1 and B_2, and each branch has two departments MD_1 and PD_1, MD_2 and PD_2 respectively. Assume that the head office has two head departments MD and PD which manage the relevant departments in both branches. Recall that a role can inherit from other roles in the RBAC system. For example, when PD has subscribed to a resource from a publisher, both PD_1 and PD_2 will have access to the resource. Similarly, a resource subscribed by the role ORG can be accessed by all the roles in the system.

By using a RBE scheme in this system, a publisher is able to encrypt the resource to the branch or department (who subscribes to the resource) and store it in the cloud so that the employees who work in the branch or department can access it. There is an assumption that these employees are trusted and will not redistribute resources of the publisher to employees who are not in that branch or department. However, it



ORG	Organisation
B1	Branch 1
B2	Branch 2
MD	Marketing Department
PD	Production Department
MD1	Marketing Department 1
PD1	Production Department 1
MD2	Marketing Department 2
PD2	Production Department 2

Figure 8.6: Digital Library System Example I

is possible that an employee leaks the content of a resource to others. Therefore, the publishers will need a trust system to assist them in identifying the roles who have malicious users, and hence avoid accepting the subscriptions from them.

Let us now consider how our trust model can be used in this system to assist the publishers (owners). Assume that no publisher has ever interacted with the role PD_1, PD_2, and now a publisher wishes to evaluate the trust of these two roles. We also assume that B_1, MD_1 and PD_1 are the same as B_2, MD_2 and PD_2 in terms of the number of employees and percentage of good feedbacks for B_1 is higher than that for B_2. Since the trust of the role is affected by descendant roles in our model, the publisher will get the result where the role PD_1 is more trusted than PD_2. This result aligns to the fact that if the branch B_1 is more trusted than the branch B_2, then the department PD_1 of the branch B_1 will also be considered more trusted than the department PD_2 of the branch B_2.

Now assume that the role ORG only has good feedbacks; that is, the resources the role ORG has subscribed have never been leaked. Since the trust value of a role is taken from the minimum value of the trust value for all its ancestor roles, when a publisher evaluates the trust of the role ORG, its trust value may be low if the good feedback percentage for B_2 is low. This is because employees in the role B_2 inherit permissions from the role ORG, and employees in this branch could potentially leak the resources.

When the role B_2 realises that its trust value is low, it may decide to warn or even exclude some potential malicious users. Then our roles' RBAC trust model can be useful to assist roles in identifying the potential malicious users. Our trust model allows the trust for employees in B_2 to be evaluated based on the feedbacks from all the roles in the organisation. That is, if an employee was working in the branch B_1 and relocated to B_2 recently, the feedbacks on the user from B_1 when the user was working in B_1 is also taken into account when B_2 determines the trustworthiness of the user.

From this digital library system example, we see that our trust model can be used in the cloud storage system using RBE schemes where role managers themselves have the flexibility in managing the user membership.

8.6 Conclusion

In this chapter, we have addressed trust issues in RBE systems for securing data storage in a cloud environment. The chapter has proposed trust models for data owners and role managers in RBAC systems which are using RBE schemes to secure stored data. These trust models assist owners and role managers to create flexible access control policies, and RBE schemes ensure that these policies are enforced in a cloud environment. The trust models enable the owners and role managers to determine the trustworthiness of individual role managers and users in the RBAC system respectively. They allow the data owners to use the trust evaluation to decide whether or not to store their encrypted data in the cloud for a particular role. The models also enable the role managers to use the trust evaluation in their decision to grant the membership to a particular user. Also, the proposed trust models take into account role inheritance and hierarchy in the evaluation of trustworthiness of roles. As far as we are aware, this is the first time such trust models for a RBAC system, taking into account role inheritance, has been proposed. Then we designed the architecture of a trust-based cloud storage system which has shown how the trust models can be integrated into a system that uses RBE schemes. We have also described the application of the trust models by considering a practical scenario and illustrating how the trust evaluations can be used to reduce the risks and enhance the quality of decision making by data owners and role managers of the cloud storage service.

9

User's Trust Model for Role-based Encryption

In this chapter, we consider the remaining two trust relationships in RBE systems, users' trust in role managers and role managers' trust in data owners. The two trust models proposed in this chapter help to improve the decision making for users and role managers in RBE systems. In a cloud storage system using RBE schemes, it is important for a user to determine whether or not a role manager in the system is trusted before joining that particular role. This would be useful especially in systems where there is a cost associated with users joining a role, for example, users need to pay a subscription fee for joining roles. When a user evaluates the trust value of a role manager, she or he may decide to proceed with joining the role only if the trust value of the role manager is above a certain trust threshold (this threshold being set by the users, and being different for different applications and context). This is achieved by our trust model for users' trust in role managers which we refer to as the User-Role RBAC trust model. Then there is the role managers' trust in data owners. In a system where data owners are allowed to choose the roles to which they can assign their data, malicious owners can cause negative behaviours of roles by deliberately assigning bad resources (e.g. virus, malware) to roles. Therefore, role managers will also need to consider the trust of the data owners to decide whether to accept data from an owner. This is achieved by our trust model for role managers' trust in owners which we refer to as the Role-Owner RBAC trust model.

These two trust models can not only prevent users from joining roles which have bad historical behaviour in terms of sharing poor quality resources or misleading users on the content of resources, but also assist the role managers to identify the malicious owners who have caused a negative impact on the roles' trustworthiness. Our users' trust model takes into account the effect of role inheritance in RBAC systems on the trust evaluation. We also present the architecture of a trust-based cloud storage system which integrates the trust models with other RBE system. Furthermore, we describe the relevance of the trust models by considering practical application scenarios and then illustrate how the trust evaluations can be used to enhance the quality of secure decision making by users and role managers of cloud storage service.

The chapter is organised as follows. Section 9.1 describes the trust issues in a cryptographic RBAC system and discusses the trust requirements for users and roles. We define the formal User-Role RBAC and Role-Owner RBAC trust models in section 9.2. The architecture of our secure cloud storage system is presented in section 9.3. In section 9.4, we illustrate how our trust models can be used in a cloud service application to enhance the quality of security decision making. Section 9.5 concludes the chapter.

9.1 Trust Issues in RBE Systems

Our proposed RBE schemes integrate cryptographic techniques with RBAC models to secure the data storage. They inherit the features and concepts from RBAC models, and also have additional components that are specific to data storage systems. In the standard RBAC model, permissions are assigned to roles by the administrator of the system. However, in a system using RBE schemes, "permissions" are the data encrypted to roles, and the security policies are specified to control the users' access to data. Because data is usually not owned by a single party, RBE systems assume that data can be encrypted to a role by whoever owns the data as opposed to the administrator in the standard RBAC system. We adopt the above described concepts for RBE systems in our trust models.

In this chapter, we assume that the data owners and users reside outside this role system infrastructure (where the roles are being administered). The issues to consider are how the users can decide whether or not to trust the role (role manager)[1] in the system and how the role managers can decide whether to trust the data owners in the system and how much to trust them. Users consider their trust in roles in order to ensure that joining roles guarantees access to data assigned to these roles. Role managers consider their trust in data owners to ensure that data owners who have assigned malicious data to the roles will not be allowed to assign data to the roles any more. In this section, we discuss the trust issues that need to be considered by the users and role managers of a RBE system.

9.1.1 Data Users' Trust in Role Managers

In some RBAC systems, user-role assignment is managed by administrators of the systems where the administrators check the qualification of users and grants role membership to them. In these systems, users trust all the roles at the same level as they are all managed by the same administrators. The roles are trusted as long as the administrators are trusted.

In RBAC systems that use the RBE schemes, users-role assignment can be decentralised to individual role managers to allow more flexibility in user management, especially in large-scale systems. Assume that in these systems users join a role based on subscription for accessing the data assigned to that role. It is clear that users need

[1] Note when we refer to role in such context we imply role managers.

to choose a trusted role when subscribing.

If the data that a user wants to access is encrypted to one role only, the user considers the trustworthiness of that role in deciding whether or not to join that role. When the same data is encrypted to multiple roles, users will need to evaluate the trustworthiness of these roles to choose the most reliable role to join. From the user's perspective, a trusted role should meet the following requirements:

- *Requirement 1: The role manager should grant membership to the users who are qualified for that role.*

  In order to access data, a user needs to join a role to which the data is encrypted. When the user requests to join the role, the role manager should give access (grant the membership) to the user if the user qualifies for that role, e.g. the user has paid the subscription fee. Refusing to give access will be considered as bad behaviour of the role manager.

- *Requirement 2: The data that a role claims to have should have been encrypted properly to that role.*

  When users want to access data, they need to know what data has been encrypted to which role so that they can choose a particular role to join. The list of the data is provided by roles. However, a user may find that she or he cannot locate or decrypt the data even after she or he has joined that specific role. This may happen if the data was not encrypted properly to that role by the owner, or the role claims to possess data that has not been encrypted to the role. Each role should take the responsibility of providing a valid and up-to-date list of the data that is in its possession.

- *Requirement 3: The data that the descendant roles of the role claim to have should have been encrypted properly to the descendant roles.*

  Since a role can inherit permissions from its descendant roles, a user who has joined a role should be able to access the data that is encrypted to any of its descendant roles. Each role is liable for the validity of the data that its descendant roles claim to have, as it is considered to be part of the data that this role has.

9.1.2 Role Managers' Trust in Data Owners

In the RBE schemes, owners can encrypt their data to any role. Obviously, role managers do not want owners to encrypt malicious data (e.g. virus, malware) to their roles. Therefore, role managers need to decide whether or not to accept data that owners want to assign to them. Having malicious data assigned to a role may result in a low trust value of the role because users who have joined the role will place negative trust records against the role if they detect that the data they get from the role is malicious. In the case where roles are profiting from users' subscriptions, low trust values in roles imply the risk of losing business.

To help role managers detect malicious owners, and hence avoid accepting data from them, another trust model is required to assist role managers in evaluating the trustworthiness of owners. Each time an owner wants to assign data to a role, the role manager will use the trust model to determine whether the data is coming from a trusted owner or not. From a role manager's perspective, a trusted owner should meet the following requirements:

- *Requirement 1: The data from the owner should be the same as its description.*

  When owners encrypt and assign data to a role, the role manager may not be able to verify each individual record from the owners. When a user who has joined a role finds that the data she or he has accessed is not the data it claims to be or contains malicious records, the user will inform the role manager about the malicious data, and the role manager should place a negative trust record against the owner who owns that data. Then next time this owner wishes to assign another data to the role, this trust record will be used by the role manager in making the decision whether or not to accept the data.

- *Requirement 2: The owner should not be considered as untrusted by any role manager in the system, if the owner has assigned data to more than one role before.*

> An owner may have had interactions with more than one role in the system. A trusted owner is supposed to act consistently in the interaction with different roles. An owner may still be considered as untrusted even though she or he has good interaction histories with a small portion of roles in the system. Therefore a trusted owner should try to maintain good interaction histories with all the roles in the system. When a role manager is interacting with an owner with which it has not interacted before, the trust opinions from the role managers of other roles can assist this role manager to determine the trustworthiness of the owner.

9.2 Trust Models for RBE systems

In this section, we consider the trust models for a RBE system. There are three types of entities in our trust models, *Owner*, *User* and *Role*. Our trust models can assist a *User* to decide whether a *Role* to interact with is trusted, and assist a *Role* in determining the trustworthiness of an *Owner*. We first review these three entities. *Owner* is the entity who owns the data and stores it in an encrypted form for particular roles in the cloud. *User* is the entity who wishes to access the data stored in the cloud. *Role* is the entity that associates users with the access to owners' data, and each role manages the user membership of itself. Here when we say that users are managed by a role, we refer to the managers of the role who determine the user set of that role.

In our trust models, we assume that all the feedback and recommendations provided are honest. In other words, we assume that the trust system has the ability to verify the submitted feedback and recommendations, and only the valid ones will be considered in the trust evaluations.

9.2.1 User-Role RBAC Trust Model

In this subsection, we consider the trust model for user's trust in roles in a RBAC system.

Definition 9.1 ***(Interaction)*** *From a user's perspective, an interaction is a transaction in which a user accesses data that is encrypted to a role to which the user belongs.*

A successful interaction is an interaction where a user has successfully accessed the data. An unsuccessful interaction is an interaction where a user failed in accessing the data to which she or he should have legitimate access. Next we define two types of unsuccessful interactions.

User Management Failure: User management failure is an unsuccessful interaction caused by incorrect user membership management of a role; that is, the role did not grant the membership to the user even when the user qualifies for the role.

Permission Management Failure: Permission management failure is an unsuccessful interaction where the data encrypted to a role is invalid, or the data is not encrypted to the role. In other words, the owner of the data did not encrypt the data to the role in question or encrypted an invalid data to the role.

Definition 9.2 ***(Trust Vector)*** *We define a trust vector to represent the behaviour history of a role as follows:*

$$\boldsymbol{v} = (r, s_U, s_P)$$

In the trust vector, r is the value related to successful interactions that users have had with a given role, s_U is the value related to *User Management Failure* of the role, and s_P is the value related to the *Permission Management Failure.*

Using the function $\mathcal{E}$ in Equation 2.1, we define the trust function $T(\boldsymbol{v})$ that represents the trust value derived from the trust vector $\boldsymbol{v}$ as

$$\mathcal{T}(\boldsymbol{v}) = \mathcal{E}(r, s_U + s_P)$$

Definition 9.3 ***(Interaction History)*** *We assume that there exists a central repository in the system that collects and stores the ratings from users on the interactions between users and roles. We define the trust record history derived from the ratings of* *the role R from n users as*

$$Hist_{\mathcal{U}}(R) = \{H_1^R, H_2^R, \cdots, H_n^R\}$$

Each entry H_i^R in $Hist(R)$ is defined as a pair of parameters, $H_i^R = \langle U_i, \boldsymbol{v}_{i,R}\rangle$, where $\boldsymbol{v}_{i,R} = (r, s_U, s_P)$ is a trust vector that represents the trust record of interactions that the user U_i has had with the role R. r is the number of U_i's positive feedbacks on the interactions with R, s_U is the number of negative feedbacks on the interactions with R due to *User Management Failure*, and s_P is the number of negative feedbacks on the interactions with R due to *Permission Management Failure*.

In a RBE system, a user who belongs to a role not only has access to the data of the role, but also has access to the data of descendent roles. Therefore, an invalid resource from a descendent role may also cause an unsuccessful interaction. Since a role knows whether a resource comes from its descendent roles, we assume that users give feedback to the roles to whom the resources are directly assigned; that is, if a user detects an invalid resource from a descendent role, she or he will update the feedback for the descendent role directly instead of the role she or he belongs to.

As discussed in section 9.1.1, from the users' perspective, the trustworthiness of a role is affected by the interaction history of the role and its descendant roles. Therefore users need to consider the following types of trust classes when evaluating the trustworthiness of roles.

Individual Trust: Individual trust is a belief that is derived directly from the interaction history of the role R.

When a user U_k wishes to evaluate the trust value of a role R, the user first obtains the interaction history $Hist_{\mathcal{U}}(R)$ of the role from the central repository. Assume that w_u is the weight that the user U_k assigns to the feedbacks from other users. Then the individual trust value of the role R can be computed as follows,

$$T_{\mathcal{U}}(R)^D = \mathcal{T}(\boldsymbol{v}_{k,R}^D),$$

$$\text{where} \quad \boldsymbol{v}_{k,R}^{D} = \boldsymbol{v}_{k,R} + w_u \cdot \sum_{i=1, i \neq k}^{n} \boldsymbol{v}_{i,R}$$

where the trust vector $\boldsymbol{v}_{k,R}^{D}$ is a combination of all trust vectors in $Hist_{\mathcal{U}}(R)$ considering the weighting for the trust vectors from other users.

Inheritance Trust: Inheritance trust is a belief that is derived from the interaction history of the descendant roles of a given role.

Assume a role R has m immediate descendant roles $\{R_1, \cdots, R_m\}$, and a weight vector $\boldsymbol{w}_{R_i}$ is defined as $(w_{R_i}^{R}, 0, w_{R_i}^{R})$ where $w_{R_i}^{R} \in [0,1]$ is the weight assigned to the inheritance relationship between R and R_i. The second element is set to zero because *User Management Failure* is not considered in inheritance trust as user management of descendant roles will not cause any unsuccessful interaction for this role. The inheritance trust of roles in a hierarchy is computed as follows:

$$T_{\mathcal{U}}(R)^{I} = \mathcal{T}(\boldsymbol{v}_{k,R}^{I}),$$

$$\text{where} \quad \boldsymbol{v}_{k,R}^{I} = \sum_{i=1}^{m} [(\boldsymbol{v}_{k,R_i}^{D} + \boldsymbol{v}_{k,R_i}^{I}), \boldsymbol{w}_{R_i}]$$

In the above equation, $[\boldsymbol{v}, \boldsymbol{w}] := \boldsymbol{v}^{T}\boldsymbol{w}$ is the usual dot product on $\mathbb{Z}_q^3$.

Combination Trust: To compute the trust value of a role, we define a combination trust function for a role R as $T_{\mathcal{U}}(R)$ to combine the above described two types of trust together. Assume that $w \in [0,1]$ is the weight of the inheritance trust. The trust value is computed as

$$T_{\mathcal{U}}(R) = (1 - w) \cdot T_{\mathcal{U}}(R)^{D} + w \cdot T_{\mathcal{U}}(R)^{I}$$

9.2.2 Example of User-Role RBAC Trust Model

Now we use an example to show how the users' trust in a role is affected by feedback for different roles in a RBAC system. In this example, we consider all the bad feedback as *Permission Management Failure*, as our intention is to show how the role hierarchy affects the trust value of roles. Consider the role hierarchy example shown in Figure

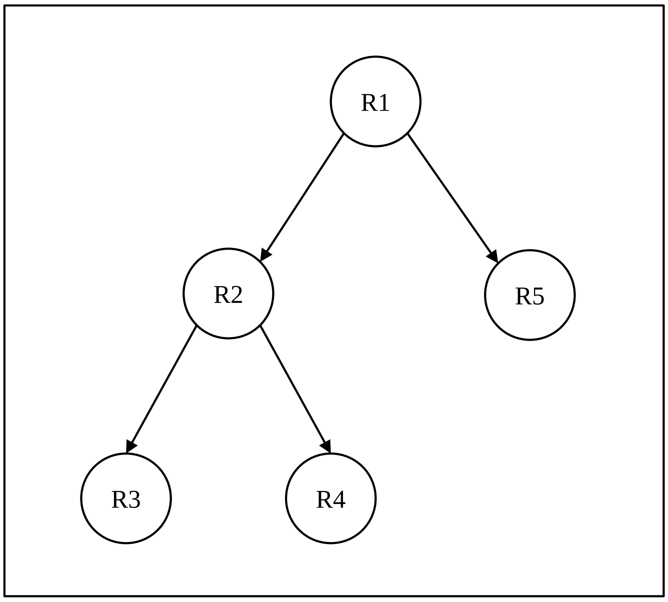


Figure 9.1: Hierarchical RBAC Example VI

9.1.

In Figure 9.1, the role R_1 inherits from role R_2 and role R_5, and the role R_2 inherits from R_3 and R_4. We set the weight between every two roles and the weight of other owners' feedback to 1; that is, the weight vector for each role R_k where $k \in [1, 5]$ is defined as $\boldsymbol{w}_{R_i} = (1, 0, 1), \forall\ i \in [1, 5], i \neq k$, and $w_u = (1, 1, 1)$. When a user wants to access a resource that has been assigned to the role R_2, she or he will need to evaluate the trust value of R_2 to decide whether R_2 is reliable to join. In Figure 9.2, we show the trust values of R_2 when only different individual roles in the RBAC system have feedback. For example, the curve for $R_1, GFP = 75\%$ shows the trust values of R_2 when only R_1 in the RBAC system has feedback, and 75% of the feedback is positive.

When the good feedback percentage is 75%, the trust value for R_2 goes up with the increasing number of feedbacks that R_2 and R_3 have. This trend implies that the more resources a role has, the more impact the good feedback percentage has on the trust value of the role. Note that the feedback for R_1 does not affect the trust value of R_2. This is because an untrusted R_1 will not cause an unsuccessful interaction of R_2. When the feedback is only given for R_2, the increase in the trust value is the fastest. This is because the individual trust of the role has more weight than the inheritance trust

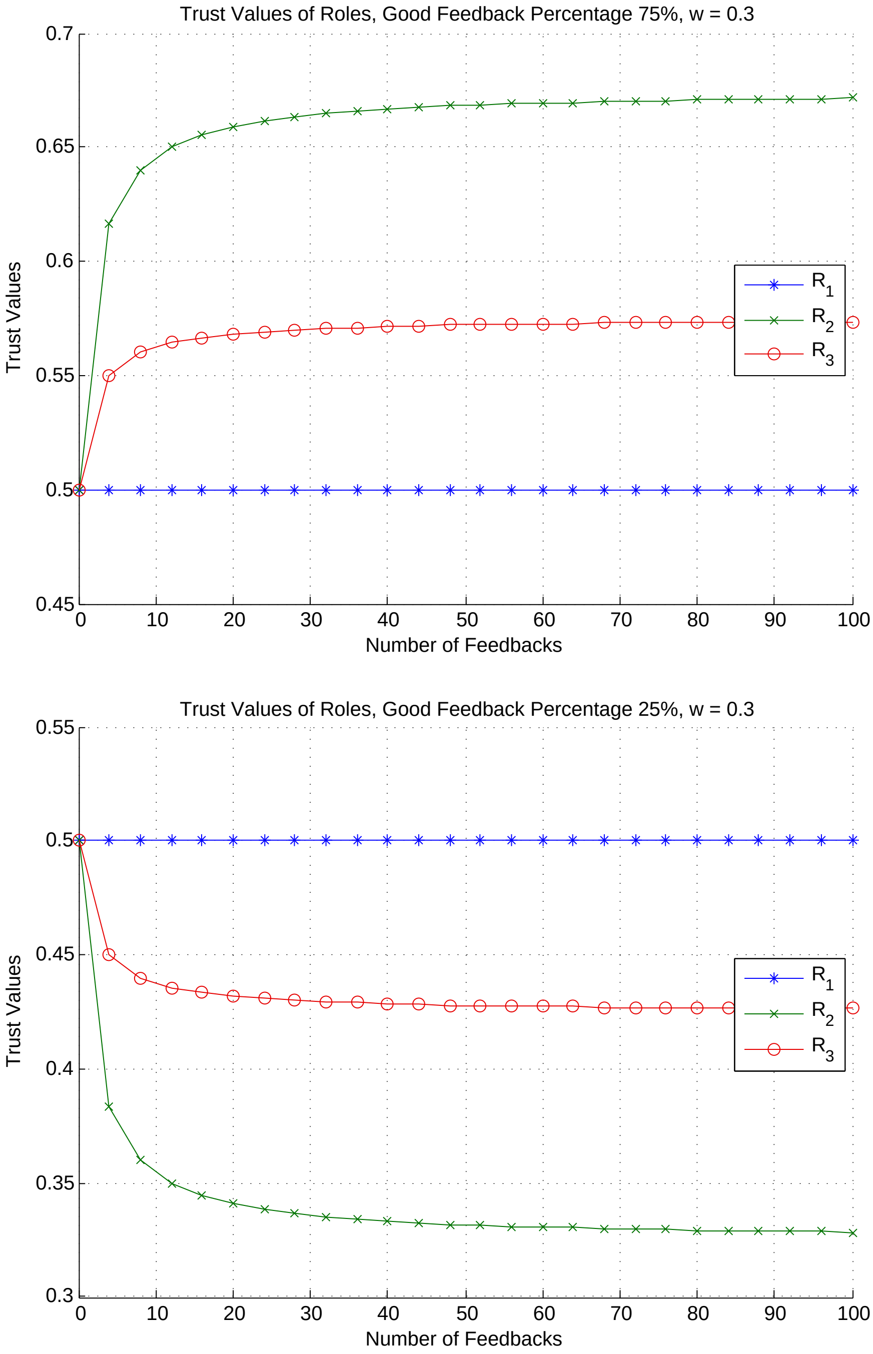


Figure 9.2: Trust Values for R_2 Evaluated by Users

by our assumption. It is clear that the increase in the trust value of R_2 is slower when the feedback is for R_3 only, because inheritance trust has less weight in this example.

When the good feedback percentage is 25%, the trust value for R_2 goes down with the increasing number of feedbacks that R_2 and R_3 have. Similarly, this trend implies that the more resources a role has, the more impact the good feedback percentage has on the trust value of the role. The feedback for R_1 does not affect the trust value of R_2 either. When feedback is only given for R_2, the decrease in the trust value is the fastest. This is because the individual trust of the role has more weight than the inheritance trust by our assumption. Therefore the decrease in the trust value of R_2 is slower when the feedback is for R_3 only.

From Figure 9.2, we see that the feedbacks for different roles in the system have different impacts on the trust value of R_2. Firstly, the feedback for ancestor roles does not affect the trust of the role. Secondly, the more resources that have been assigned to a role, the more impact the feedback for the role will have on its ancestor roles as well as itself. These results show that our users' trust model is useful in assisting users to determine properly the trust of roles in RBAC systems.

9.2.3 Role-Owner RBAC Trust Model

In the case when any owner can choose roles to encrypt their resources to, assigning malicious resources or invalid resources to a role may cause the *Permission Management Failure* of the role. Therefore, it would be useful to have a trust model to assist role managers in determining the trustworthiness of an owner, and hence decide whether or not to accept the resources from the owner.

As discussed in section 9.1.2, the trust requirement on owners is simpler comparing to the users' trust in roles, and we see some important differences. The trust in owners is independent from the role hierarchy; that is, the role hierarchy does not affect the trustworthiness of owners. We note that a general trust model can be used in this scenario. For completeness purposes, we also give the definition of the trust model for the role managers' trust in data owners in this subsection.

Definition 9.4 ***(Interaction)*** *From a role manager's perspective, an interaction with an owner is a transaction in which an owner assigned a resource to that role, and that the role manager has accepted the resource.*

Definition 9.5 ***(Trust Vector)*** *We define a trust vector to represent the behaviour* *history of an owner as follows:*

$$\boldsymbol{v} = (h, s)$$

where h is the value related to resources owned by the owner, and s is the value related to malicious or invalid resources owned by the owner.

Using the function $\mathcal{E}$ in Equation 2.1, we define the trust function $T(\boldsymbol{v})$ that represents the trust value derived from the trust vector $\boldsymbol{v}$ as

$$\mathcal{T}(\boldsymbol{v}) = \mathcal{E}(h - s, s)$$

Definition 9.6 ***(Interaction History)*** *We assume that there exists a central repository in the system that collects and stores the behaviour histories provided by role managers to which the owner has assigned the resources. We define the trust record history provided by a set $\mathcal{R}$ of n roles as*

$$Hist_{\mathcal{R}}(O) = \{H_1^O, H_2^O, \cdots, H_n^O\}$$

Each entry H_i^O in $Hist(O)$ is defined as a pair of parameters, $H_i^O = \langle R_i, v_{i,O} \rangle$ where $\boldsymbol{v}_{i,O} = (h, s)$ is a trust vector that represents the trust record of the owner O on the resources that she or he has assigned to the role R_i. h is the total number of O's resources that has been assigned to R_i, and s is the number of bad resources assigned by O.

We assume that an owner O has a resource and wants to assign it to a role R_k. When this resource is assigned to the role R_k, R_k updates the trust record of the owner by increasing the value h in the trust vector H_k^O of O by 1. Now assume that a user has found the resource to be invalid, and then she or he reports to the role of this resource. If the role has confirmed that the user's complaint is true after verifying the resource, R_k will find out that it is O who uploaded this resource, and R_k will increase the value s in trust vectors H_k^O for this owner by 1.

A user that belongs to a role has the permission to access resources of the descendant roles of the role. When the user reports a bad resource from its descendant role, this role may not be able to identify the owner of the resource as the resource is not assigned to this role directly. Hence the role cannot update the trust records of the owner. In this case, the role can notify all its descendant roles about this bad resource, and the role to which the resource is assigned to will update the trust record of the owner who owns the resource.

Assume that w is the weight that the role R_k assigns to the feedback from other roles. Taking as input the interaction history of an owner, the trust value of the owner can be computed as follows:

$$T_{\mathcal{R}}(O) = \mathcal{T}(\boldsymbol{v}_{k,O}^{T}), \quad \boldsymbol{v}_{k,O}^{T} = \boldsymbol{v}_{k,O} + w \cdot \sum_{i=1, i\neq k}^{n} \boldsymbol{v}_{i,O}$$

This trust value is evaluated based on a combination of all trust records in $Hist_{\mathcal{R}}(O)$ considering the weighting for the trust records from other roles.

9.3 Architecture

In this section, we present the design of a secure cloud storage system by combining the trust models for RBAC proposed in section 9.2 with a RBE system. This architecture provides a practical solution for building a reliable and trusted RBAC system while retaining the use of cryptographic techniques.

9.3.1 System Overview

Consider the system architecture shown in Figure 9.3. Each solid line in the figure shows the communication channel set by the system between two components joined together by the line, and the arrows indicate the direction in which the information flows. Since our trust models are based on RBE schemes, our system contains all the entities that a RBE scheme has, including an administrator, roles, users, and owners. The administrator is the system administrator of the RBAC system, and it generates the system parameters and issues all the necessary credentials. In addition, the administrator manages the role hierarchy of the system. To put a role into the role hierarchy, the administrator needs to compute the parameters for that role. These parameters represent the position of the role in the role hierarchy. They are stored in the cloud, and are available publicly. Roles are the entities that associate users and owners together. Each role has its own role parameters which define the user membership. These role parameters are stored in the cloud, and a role needs to update them in the cloud when updating the user membership of the role. Owners are the parties who possess the data and want to store the encrypted data in the cloud for other users to access, and they specify the roles who can access the data. In the RBAC model, they are the parties who manage the relationship between permissions and roles. Users are the parties who wish to acquire certain data from the cloud. When a user wishes to access stored data in the cloud, she or he first sends the request to the cloud, and decrypts the data upon receiving the response from the cloud.

In addition to these four entities in a basic RBE scheme, our trust models enhanced RBE system by integrating an extra trust management system, which consists of five components. Next, we describe the details of these components.

Central Repository : In our trust models, all the interaction histories and trust records related to roles and users are stored in a central repository. The central repository is used to keep the records of all these interaction histories and trust records which are used by the Trust Decision Engine (described below) in evaluating the trust value of roles and owners. Any entity that is residing outside the trust management system is not able to access the central repository.

Role Behaviour Auditor : In order to protect the integrity of the feedback on roles, a role behaviour auditor collects the feedback for roles from users. The role behaviour auditor needs to ensure that a user who uploads feedback against a role has been granted the membership of the role or an ancestor role of that role. All the valid feedback will be forwarded to the central repository, and invalid feedback will be discarded.

Owner Behaviour Auditor : An owner behaviour auditor is an entity to collect the feedback on owners' behaviour. However, unlike the role behaviour auditor, the owner behaviour auditor listens for feedback on two channels. One is from the roles who

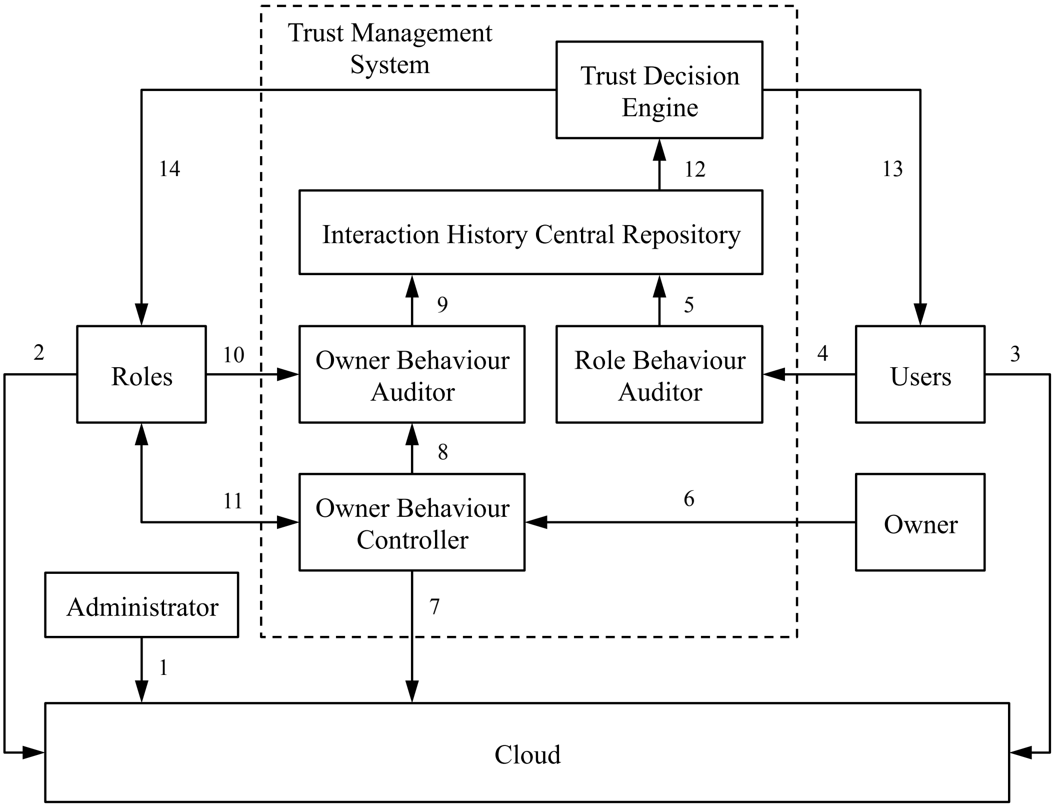


Figure 9.3: Architecture for Using Users' Trust Models in a RBE System

may report the invalid data, and another is from the owner behaviour controller which reports the ownership of the stored data in the cloud. This auditor will determine whether an owner has uploaded any malicious or invalid data to the cloud, and can update the central repository.

Owner Behaviour Controller : Owner behaviour controller acts as a proxy server between owners and the cloud. It controls and forwards the owners' encrypted data to the cloud. The controller can decide whether to store data in the cloud based on the decision from the role to which the data is assigned. The controller will inform the owner behaviour auditor about which owner the uploaded data belongs to.

Trust Decision Engine : The trust decision engine is the entity which evaluates the trust of the roles for users and the trust of the owners for roles. The trust decision engine takes as input the interaction histories or trust records stored in the central repository, and outputs the trust value of a particular role or owner.

9.3.2 System Workflow

All the entities in the system are connected through different communication channels which are labelled with numbers in Figure 9.3. We explain how the system works by describing the information flow through these channels.

First, the administrator initialises the system and specifies the role hierarchy of the system. The generated system parameters are uploaded to the cloud via channel 1. Roles grant the membership to users, and upload role parameters to the cloud via channel 2. Users download and decrypt data from the cloud via channel 3. When an owner wants to encrypt and store data in the cloud to a particular role, she or he first encrypts the data and sends a request to the owner behaviour controller via channel 6. Then the owner behaviour controller notifies the role via channel 11 and forwards the request to the cloud through channel 7 if the role agrees to accept the data from this owner. The cloud then communicates with the owner as in a normal RBE scheme. The controller also sends the owner behaviour auditor the information about the owner's identity and the resource's identity via channel 8.

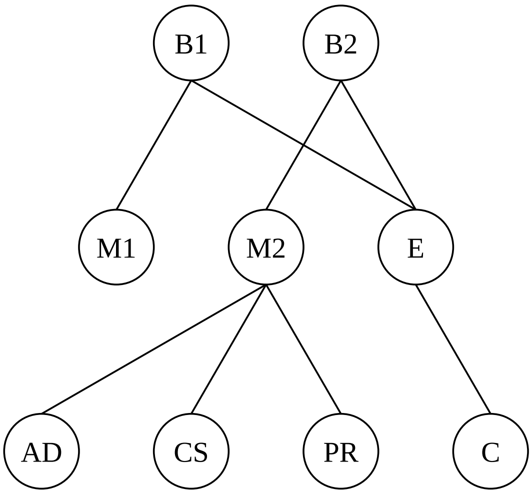
When a user wants to access a resource in the RBAC system, the system first returns a list of roles who claim to have this resource. Then the user requests the trust evaluation on these roles from the trust management system. The trust value of the roles will be returned to the user through the channel 13. The user may choose a role who has the highest trust value to send the join request. When a user has found that the data she or he has accessed from the role is malicious or invalid, she or he then provides feedback on the role to whom the resource is encrypted to the role behaviour auditor through channel 4. Once the role behaviour auditor verifies that the feedback is from an authorised user, it will forward the feedback to the central repository.

When a negative feedback of a role has been raised by a user because of an invalid resource, the role will send the identity of the resource to the owner behaviour auditor via channel 10 if it believes that the resource was invalid when the owner uploaded the resource. The auditor then updates the trust records of the owner of this resource to the central repository via channel 9. When an owner wants to assign a resource to a role, the role can ask the trust management system about the trust evaluation for an owner, and the trust value will be returned by the trust decision engine through channel 14. Upon receiving the trust values for the owner from the trust decision engine, the role can inform the owner behaviour controller via channel 11 whether to accept the data. Moreover, this trust evaluation process can be made automatically by connecting the owner behaviour controller to the trust decision engine directly. Roles can pre-determine a trust threshold for accepting data from owners. Every time an owner wants to upload a resource, the owner behaviour controller can check the trust value of the owner from the trust decision engine directly, and decide whether to accept the resource by comparing the trust value with the role's threshold.

9.4 Application Scenario

In this section, we describe a digital library system which uses our proposed trust models to illustrate how the trust models can assist the security decision making in this system. The digital library system uses an external public cloud to store all the digital format resources such as books, papers, theses, and other types of publications. There are many distributors who use the platform provided by the digital library system to share digital resources. Each distributor can get the authorisations for sharing the digital resources from the publishers directly. A party who subscribes to a distributor can access all the resources of the distributor. Assume that the distributors have two types of subscription licenses; personal licenses that allow only the subscribed user to access the resources, and business licenses that allow another distributor to resell the resources to other users or distributors.

Now let us consider the example of a distributors' network for this digital library system. The hierarchical relationship of the distributors is shown in Figure 9.4. In this system, distributors choose the resources to share by their categories. The distributors AD, CS, PR, C, and M_1 get the authorisations for selling digital resources in the categories *Advertising*, *Customer Service*, *Public Relations*, *Commerce* and *Marketing* respectively from the publishers. Distributors M_2 and E sell a wider ranger of resources which cover all the categories in *Marketing* and *Economics* respectively,



B1	Business 1
B2	Business 2
M1	Marketing 1
M2	Marketing 2
E	Economics
AD	Advertising
CS	Customer Service
PR	Public Relations
C	Commerce

Figure 9.4: Digital Library System Example II

and these two distributors get authorisations from the distributors of sub-categories instead of the publishers directly. Note that the categories of resources sold by M_1 and M_2 are overlapped. The difference is the channels they get the resources from: M_1 from publishers, and M_2 from sub-distributors. Similarly, distributors B_1 and B_2 get authorisations from M_1, E, and M_2, E respectively, and their resources both cover the categories *Business*.

To use RBE schemes to protect the resources so that only the authorised users can access them, the administrator of the digital library system first sets up the system parameters based on the relationships of the distributors. Then the publishers can encrypt their resources to the distributors whom they authorised to sell the resources. Here we consider the distributors as roles in the RBAC, and publishers as owners of the resources. When a user subscribes to a distributor, the distributor simply adds the user to the role. Then the user can use the key given by the system administrator to decrypt the resources of the role. Because the RBE schemes support role hierarchy, in this example, users who subscribed to the role M_2 can also access the resources of the role AD, CS and PR, and users subscribed to B_1 can access the resources of all the roles M_1, E, and C.

First let us consider how the trust model can assist the users. Assume that the distributor M_2 also gets some resources, which the distributors AD, CS, and PR do not have directly from the publishers. To save the cost of storing resources in the cloud, M_2 chooses to reprint some resources in a lower quality to reduce the file size. Users subscribed to M_2 may give negative feedbacks on M_2 because they have difficulties in reading some of the resources. Later on, when a user want to access marketing resources, she or he evaluates the trust of M_1 and M_2, and the trust model will output a higher trust value for M_1 than for M_2 because of the negative feedbacks of M_2. Then the user will know the quality of resources from M_1 is better than those from M_2. However, the distributors AD, CS, and PR will not be affected because the poor quality resources are not coming from them. When a user wants to subscribe to a distributor for $Business$, B_2 will have lower trust value than B_1 as resource the user would get from B_1 may come from M_2.

Now let us look at the trust model for roles' trust in owners. Assume that publishers want to promote their digital resources, and they actively assign their resources to distributors. The resources that have come from some publishers may be of poor quality or alternatively some resources are not what the publishers claim them to be. The distributors may not be able to verify each individual resource due to the lack of expertise in certain areas. When users complain about a bad resource, the role can give a negative feedback on the publisher who owns the resource, after confirming that the users' complaints is valid. The feedback of the publisher can be accessed by all the distributors so they can avoid using this publisher in the future.

9.5 Conclusion

In this chapter, we have addressed trust issues in RBE systems for securing data storage in a cloud environment. We have proposed trust models for users and role managers in RBAC systems which are using RBE schemes to secure stored data. These trust models assist the users and role managers to determine the trustworthiness of individual roles and owners in the RBAC system respectively. They allow the users to perform the trust evaluation to decide whether or not to access a resource from a particular role. Our trust model takes into account role inheritance and hierarchy in the evaluation of trustworthiness of roles. The models also enable the role managers to use the trust evaluation in their decision to accept the resources from a particular owner. We have given the design of an architecture of a trust-based cloud storage system which has integrated these trust models with the RBE schemes. We have also described the application of the proposed trust models by considering a practical scenario and illustrating how the trust evaluations can be used to reduce the risks and enhance the quality of security decision-making by users and role managers of the cloud storage service.

The proposed trust models use a centralised trust management system to assist users and role managers with their trust evaluations. Though the users and role managers in the system still need to trust the centralised trust management components, we believe that this approach has improved the cases where users and role managers need to trust each individual roles and data owners in the system. We note that the auditing components in our designed architecture need to collect all the provided feedback. In large-scale systems, the load of these auditing components could be high. One solution to this issue involves the use of decentralised auditing components which will be considered in our future work. In addition, we only considered two types of feedbacks in our trust models, positive and negative. However, a user who has unsatisfactory experiences with roles may want to provide varying levels of negative feedback. For example, one user may have retrieved a malware instead of valid data from a role, where as another one might have obtained poor quality data instead of good quality data from the same role. It is clear that the latter case is less harmful than the former one, and the user may give a lower degree of negative feedback in the former case.

10

Conclusions and Future Work

In this chapter, we summarise the main contributions of this thesis, and discuss some issues that may be addressed in future work.

10.1 Contributions

In this thesis, we have proposed a novel approach to secure the data stored in a cloud system using a combination of cryptographic techniques and RBAC models to manage the policies that control access to the stored data. First we proposed new role-based encryption (RBE) schemes which ensure that the stored data in the cloud is only accessible to those users who satisfy the role-based access policies. The role-based policies are specified by the data owners, which can be individuals or enterprises. Then we developed the system architecture of the proposed secure RBE-based data storage system using a hybrid cloud infrastructure. The data is stored in an encrypted form in the public cloud whereas the role-based policies and hierarchy reside in a private cloud. Such hybrid architecture reflects the practical environment. Then we described a cryptographic administrative RBAC model to integrate our proposed RBE schemes with administrative RBAC models to achieve flexible and efficient administration in large-scale systems using the proposed RBE schemes. We demonstrated the applicability of the proposed architecture and the RBE secure system in two realistic scenarios, namely a banking system storing its documents in a cloud and the secure storage of electronic patient records in a cloud-based storage system. Finally, the thesis analysed trust issues in such a secure RBE-based cloud data storage system and developed new trust models to assist not only role managers to identify untrusted data owners and users but also data owners and users to determine the trust of the role managers. The latter is significant as there is no existing trust model which considers the trust of the RBAC system itself. Our proposed trust models take into account the role hierarchy and inheritance in the trust evaluation.

- We started with an analysis of the security requirements in cloud storage systems using a cryptographic approach to protect the data. As RBAC is commonly used to control access to data, we decided to develop new schemes that can integrate secure cryptographic techniques with RBAC models to protect data stored in the cloud. To improve the flexibility and the efficiency of cryptographic RBAC schemes, we proposed and formalised a new type of cryptographic RBAC scheme which we called role-based encryption (RBE) and defined the security properties of the new RBE scheme.

- Three generic constructions for RBE schemes have been proposed to show that RBE schemes can be constructed by using secure ID-based broadcast encryption (IBBE) schemes. We developed a framework which described how IBBE techniques could be used to build RBE schemes with different features. We proposed three RBE constructions, compared the differences between them, and discussed suitable scenarios for using these individual constructions depending on each scheme's advantages.

- A specific RBE scheme has been proposed using a broadcast encryption mechanism described in [48]. Compared to other existing cryptographic RBAC schemes, our scheme has several superior characteristics, such as efficient user management and constant size ciphertext and decryption keys. Then we proposed an improved RBE scheme which has an efficient user revocation. A significant characteristic of the improved RBE scheme is that revocation of a user from a role does not affect other users and roles in the system.

- A secure cloud data storage architecture was then designed based on a hybrid cloud infrastructure using the improved RBE scheme. In this architecture, the public cloud is used to store only the encrypted data, whereas the organisation's sensitive structure information such as the role hierarchy and user membership information is stored in a private cloud to which only the administrators and role managers have access. Such hybrid architecture helps to reduce the attack surface.

- A complete secure cloud storage system was then developed using the improved RBE scheme and the hybrid cloud architecture. This system adopted the strategy of outsourcing part of the computation to the cloud to achieve efficient client operations in the implementation. We also optimised the implementation of the decryption algorithm and showed that the cloud's decryption time can be reduced by using the parallel computing techniques on the server side.

- To simplify the administration tasks in large-scale RBAC systems, we proposed a cryptographic administrative model AdC-RBAC. The AdC-RBAC model uses cryptographic techniques to manage and enforce administrative policies for RBE schemes. This model allows only the authorised administrative roles to change RBAC system access policies. We have described three components of this model: UAM for user membership management, PAM for permission management, and

RAM for role management.

- Two application scenarios of using the proposed secure RBE-based system have been discussed. They show how our secure system can be used to protect data in cloud storage systems. We have described a banking application in which the RBE scheme is used for document sharing and distribution in a cloud platform. The second application is a cloud storage system for securely storing the patient-centric patients' health records (PHR); we have given a detailed design of the PHR structure which allows the patients to have flexible control over their PHR stored in the cloud system.
- Then we analysed the trust issues in RBE-based cloud data storage systems and proposed trust models to reason about and enhance the security of data stored in the cloud. The owners' trust models assist the owners and roles to determine the trustworthiness of individual roles and users respectively in the RBAC system, and the users' trust models assist the users and roles in determining the trustworthiness of individual roles and owners respectively.

The proposed security techniques in this thesis constitute an integrated and comprehensive secure cloud data storage system. In this thesis, we have provided the designs of schemes and architecture to show how such an integrated system can be achieved in practice in a large-scale system, without losing the features and advantages of each of the individual components.

10.2 Future Work

In this section, we briefly outline some areas of further work that are worth investigating in the future, as a follow on from the work done in this thesis.

The first area of work worth considering is related to searchable schemes on encrypted data. In our system, the data is stored in the cloud in the form of key-value pairs, and we assume that users know the key (index) of the data that they want to access. When users want to access data from the cloud, they need to enter the index of the data in their requests to the cloud system. In a more general scenario, the ability to search is desirable, which allows a user to search for a particular resource using keywords or names. There are many searchable encryption schemes in the literature which allow users with valid decryption keys to make search queries on encrypted data without leaking information through the queries or the data. However, there is no existing searchable encryption scheme that can work directly with RBAC models. Hence, a solution that allows users to search over the data that is encrypted using RBE schemes without violating the access control policies would definitely result in a better user experience. With such a solution, users should be allowed to query a keyword or a name to the cloud, and only the encrypted data that matches the query and the access policies will be returned by the cloud to the users. The cloud should not learn anything about the queries or the data.

Another area of further work could be to add the anonymity feature to our secure RBE-based cloud data storage system. When a user accesses the data in a cloud system, for the cloud to make the decision on returning the encrypted data to the user, the cloud needs to know the identity of the user as well as the identity of the role that the user belongs to. In our RBE systems, these identities are considered as public parameters. However, some users may not want the cloud to know their identities and the roles that they belong to. Therefore, there may be a need for a user to send the request anonymously to the cloud, and the cloud still needs to determine whether the user has the access to the encrypted data without knowing the identity of the user or the role. In the first piece of further work mentioned above, we considered the possibility of a desirable searchable solution for RBE schemes. We can envisage combining this anonymity feature with the search scheme leading to an anonymous search solution. In such anonymous search users would be able to make search queries to the cloud anonymously, and the cloud will return the suitable encrypted data to the users without knowing the queries and the data as well as the identities of the users. Integrating such an anonymous searchable solution into our RBE system would benefit the users for allowing them to better locate and access the data in secure cloud storage systems.

Bibliography

[1] Selim G. Akl and Peter D. Taylor. “Cryptographic Solution to a Problem of Access Control in a Hierarchy”. In: *ACM Transactions on Computer Systems* 1.3 (1983), pp. 239–248.

[2] Jacob Alperin-Sheriff and Chris Peikert. “Circular and KDM Security for Identity-Based Encryption”. In: *Public Key Cryptography - PKC 2012*. Vol. 7293. LNCS. Darmstadt, Germany: Springer, May 2012, pp. 334–352.

[3] Benny Applebaum. “Key-Dependent Message Security: Generic Amplification and Completeness”. In: *Advances in Cryptology - EUROCRYPT 2011*. Vol. 6632. LNCS. Tallinn, Estonia: Springer, May 2011, pp. 527–546.

[4] Michael Armbrust et al. “A View of Cloud Computing”. In: *Communications of the ACM* 53.4 (2010), pp. 50–58.

[5] Mikhail J. Atallah, Keith B. Frikken, and Marina Blanton. “Dynamic and Efficient Key Management for Access Hierarchies”. In: *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005*. Alexandria, VA, USA: ACM, Nov. 2005, pp. 190–202.

[6] Jean-Philippe Aumasson. *On the pseudo-random generator ISAAC*. Cryptology ePrint Archive, Report 2006/438. http://eprint.iacr.org/. 2006.

[7] Avanade. *Global Survey: Has Cloud Computing Matured?* http://www.avanade.com/Documents/Research%20and%20Insights/Global_Survey_Slide_Graphics_Has_Cloud_Matured.pdf. 2011.

[8] Elaine Barker et al. *Recommendation for Key Management - Part 1: General (Revision 3).* Tech. rep. NIST, May 2011.

[9] Paulo S. L. M. Barreto and Michael Naehrig. "Pairing-Friendly Elliptic Curves of Prime Order". In: *Selected Areas in Cryptography, 12th International Workshop, SAC 2005.* Vol. 3897. LNCS. Kingston, ON, Canada: Springer, Aug. 2005, pp. 319–331.

[10] Paulo S. L. M. Barreto et al. "Efficient and Provably-Secure Identity-Based Signatures and Signcryption from Bilinear Maps". In: *Advances in Cryptology - ASIACRYPT 2005.* Vol. 3788. LNCS. Chennai, India: Springer, Dec. 2005, pp. 515–532.

[11] David Elliott Bell and Leonard J. LaPadula. *Secure computer systems: Mathematical foundations and model.* Tech. rep. M74-244. Bedford, MA: MITRE Corporation, 1975.

[12] John Bethencourt, Amit Sahai, and Brent Waters. "Ciphertext-Policy Attribute-Based Encryption". In: *IEEE Symposium on Security and Privacy (S&P 2007).* Oakland, California, USA: IEEE Computer Society, May 2007, pp. 321–334.

[13] Kenneth J. Biba. *Integrity Considerations for Secure Computer Systems.* The Mitre Corporation. Bedford, MA, Apr. 1977.

[14] John Black, Phillip Rogaway, and Thomas Shrimpton. "Encryption-Scheme Security in the Presence of Key-Dependent Messages". In: *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002.* Vol. 2595. LNCS. St. John's, Newfoundland, Canada: Springer, Aug. 2002, pp. 62–75.

[15] Matt Blaze, Joan Feigenbaum, and Jack Lacy. "Decentralized Trust Management". In: *IEEE Symposium on Security and Privacy.* Oakland, CA, USA: IEEE Computer Society, May 1996, pp. 164–173.

[16] Matt Blaze et al. *The KeyNote Trust-Management System Version 2.* Tech. rep. Internet Society, Network Working Group, 1999.

[17] Carlo Blundo et al. “Efficient Key Management for Enforcing Access Control in Outsourced Scenarios”. In: *24th IFIP TC 11 International Information Security Conference, SEC 2009*. Vol. 297. IFIP Advances in Information and Communication Technology. Pafos, Cyprus: Springer, May 2009, pp. 364–375.

[18] Dan Boneh and Xavier Boyen. “Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles”. In: *Advances in Cryptology - EUROCRYPT 2004*. Vol. 3027. LNCS. Interlaken, Switzerland: Springer, May 2004, pp. 223–238.

[19] Dan Boneh and Xavier Boyen. “Secure Identity Based Encryption Without Random Oracles”. In: *Advances in Cryptology - CRYPTO 2004*. Vol. 3152. LNCS. Santa Barbara, California, USA: Springer, Aug. 2004, pp. 443–459.

[20] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. “Hierarchical Identity Based Encryption with Constant Size Ciphertext”. In: *Advances in Cryptology - EUROCRYPT 2005*. Vol. 3494. LNCS. Aarhus, Denmark: Springer, May 2005, pp. 440–456.

[21] Dan Boneh and Matthew Franklin. “Identity-Based Encryption from the Weil Pairing”. In: *SIAM Journal on Computing* 32.3 (2003), pp. 586–615.

[22] Dan Boneh, Craig Gentry, and Brent Waters. “Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys”. In: *Advances in Cryptology - CRYPTO 2005*. Ed. by Victor Shoup. Vol. 3621. LNCS. Santa Barbara, California, USA: Springer, Aug. 2005, pp. 258–275.

[23] Dan Boneh and Michael Hamburg. “Generalized Identity Based and Broadcast Encryption Schemes”. In: *Advances in Cryptology - ASIACRYPT 2008*. Vol. 5350. LNCS. Melbourne, Australia: Springer, Dec. 2008, pp. 455–470.

[24] Dan Boneh and Jonathan Katz. “Improved Efficiency for CCA-Secure Cryptosystems Built Using Identity-Based Encryption”. In: *Topics in Cryptology - CT-RSA 2005, The Cryptographers’ Track at the RSA Conference 2005*. Vol. 3376. LNCS. San Francisco, CA, USA: Springer, Feb. 2005, pp. 87–103.

[25] Dan Boneh and Brent Waters. “Conjunctive, Subset, and Range Queries on Encrypted Data”. In: *4th Theory of Cryptography Conference, TCC 2007.* Vol. 4392. LNCS. Amsterdam, The Netherlands: Springer, Feb. 2007, pp. 535–554.

[26] Dan Boneh et al. “Circular-Secure Encryption from Decision Diffie-Hellman”. In: *Advances in Cryptology - CRYPTO 2008.* Vol. 5157. LNCS. Santa Barbara, CA, USA: Springer, Aug. 2008, pp. 108–125.

[27] Dan Boneh et al. “Public Key Encryption with Keyword Search”. In: *Advances in Cryptology - EUROCRYPT 2004.* Vol. 3027. LNCS. Interlaken, Switzerland: Springer, May 2004, pp. 506–522.

[28] BouncyCastle. *Bouncy Castle Cryptography Library.* http://www.bouncycastle.org/.

[29] Xavier Boyen and Brent Waters. “Anonymous Hierarchical Identity-Based Encryption (Without Random Oracles)”. In: *Advances in Cryptology - CRYPTO 2006.* Vol. 4117. LNCS. Santa Barbara, California, USA: Springer, Aug. 2006, pp. 290–307.

[30] D. F. C. Brewer and M. J. Nash. “The Chinese Wall Security Policy”. In: *Proceedings of the 1989 IEEE Symposium on Security and Privacy.* Oakland, California, USA: IEEE Computer Society, May 1989, pp. 206–214.

[31] Jan Camenisch, Nishanth Chandran, and Victor Shoup. “A Public Key Encryption Scheme Secure against Key Dependent Chosen Plaintext and Adaptive Chosen Ciphertext Attacks”. In: *Advances in Cryptology - EUROCRYPT 2009.* Vol. 5479. LNCS. Cologne, Germany: Springer, Apr. 2009, pp. 351–368.

[32] Jan Camenisch and Anna Lysyanskaya. “An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation”. In: *Advances in Cryptology - EUROCRYPT 2001.* Vol. 2045. LNCS. Innsbruck, Austria: Springer, May 2001, pp. 93–118.

[33] Ran Canetti, Shai Halevi, and Jonathan Katz. "Chosen-Ciphertext Security from Identity-Based Encryption". In: *Advances in Cryptology - EUROCRYPT 2004*. Vol. 3027. LNCS. Interlaken, Switzerland: Springer, May 2004, pp. 207–222.

[34] Angelo De Caro and Vincenzo Iovino. *Java Pairing Based Cryptography Library.* http://libeccio.dia.unisa.it/projects/jpbc/.

[35] Jae Choon Cha and Jung Hee Cheon. "An Identity-Based Signature from Gap Diffie-Hellman Groups". In: *Public Key Cryptography - PKC 2003*. Vol. 2567. LNCS. Miami, FL, USA: Springer, Jan. 2003, pp. 18–30.

[36] Sudip Chakraborty and Indrajit Ray. "TrustBAC - Integrating Trust Relationships into the RBAC Model for Access Control in Open Systems". In: *11th ACM Symposium on Access Control Models and Technologies, SACMAT 2006*. Lake Tahoe, California, USA: ACM, June 2006, pp. 49–58.

[37] Melissa Chase. "Multi-authority Attribute Based Encryption". In: *4th Theory of Cryptography Conference, TCC 2007*. Vol. 4392. LNCS. Amsterdam, The Netherlands: Springer, Feb. 2007, pp. 515–534.

[38] Melissa Chase and Sherman S. M. Chow. "Improving Privacy and Security in Multi-Authority Attribute-Based Encryption". In: *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009*. Chicago, Illinois, USA: ACM, Nov. 2009, pp. 121–130.

[39] Ling Cheung and Calvin Newport. "Provably Secure Ciphertext Policy ABE". In: *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007*. Alexandria, Virginia, USA: ACM, Oct. 2007, pp. 456–465.

[40] Hung-Yu Chien and Jinn-ke Jan. "New Hierarchical Assignment without Public Key Cryptography". In: *Computers & Security* 22.6 (2003), pp. 523–526.

[41] Clifford Cocks. "An Identity Based Encryption Scheme Based on Quadratic Residues". In: *Cryptography and Coding, 8th IMA International Conference.* Vol. 2260. LNCS. Cirencester, UK: Springer, Dec. 2001, pp. 360–363.

[42] Jason Crampton. “Understanding and Developing Role-Based Administrative Models”. In: *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005*. Alexandria, VA, USA: ACM, Nov. 2005, pp. 158–167.

[43] Jason Crampton and George Loizou. “Administrative Scope: A Foundation for Role-Based Administrative Models”. In: *ACM Transactions on Information and System Security* 6.2 (2003), pp. 201–231.

[44] Jason Crampton and George Loizou. “Administrative Scope and Role Hierarchy Operations”. In: *7th ACM Symposium on Access Control Models and Technologies, SACMAT 2002*. Monterey, California, USA: ACM, June 2002, pp. 145–154.

[45] Jason Crampton, Keith M. Martin, and Peter R. Wild. “On Key Assignment for Hierarchical Access Control”. In: *19th IEEE Computer Security Foundations Workshop, CSFW 2006*. Venice, Italy: IEEE Computer Society, July 2006, pp. 98–111.

[46] Isabel F. Cruz et al. “A Constraint and Attribute Based Security Framework for Dynamic Role Assignment in Collaborative Environments”. In: *Collaborative Computing: Networking, Applications and Worksharing, 4th International Conference, CollaborateCom 2008*. Vol. 10. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Orlando, FL, USA: Springer, Nov. 2008, pp. 322–339.

[47] Department of Defense. *Trusted Computer System Evaluation Criteria*. DOD 5200.28-STD (supersedes CSC-STD-001-83). Department of Defense. Dec. 1985.

[48] Cécile Delerablée. “Identity-Based Broadcast Encryption with Constant Size Ciphertexts and Private Keys”. In: *Advances in Cryptology - ASIACRYPT 2007*. Vol. 4833. LNCS. Kuching, Malaysia: Springer, Dec. 2007, pp. 200–215.

[49] Cécile Delerablée, Pascal Paillier, and David Pointcheval. “Fully Collusion Secure Dynamic Broadcast Encryption with Constant-Size Ciphertexts or Decryption Keys”. In: *Pairing-Based Cryptography - Pairing 2007*. Vol. 4575. LNCS. Tokyo, Japan: Springer, July 2007, pp. 39–59.

[50] Yahalom Klein Delta et al. “Trust Relationships in Secure Systems-A Distributed Authentication Perspective”. In: *In Proceedings, IEEE Symposium on Research in Security and Privacy*. Oakland, CA, USA, May 1993, pp. 150–164.

[51] Dorothy E. Denning. “A Lattice Model of Secure Information Flow”. In: *Commun. ACM* 19.5 (1976), pp. 236–243.

[52] Keita Emura et al. “A Ciphertext-Policy Attribute-Based Encryption Scheme with Constant Ciphertext Length”. In: *Information Security Practice and Experience, 5th International Conference, ISPEC 2009*. Vol. 5451. LNCS. Xi’an, China: Springer, Apr. 2009, pp. 13–23.

[53] Fujun Feng et al. “A Trust and Context Based Access Control Model for Distributed Systems”. In: *10th IEEE International Conference on High Performance Computing and Communications, HPCC 2008*. Dalian, China: IEEE, Sept. 2008, pp. 629–634.

[54] David F. Ferraiolo and D. Richard Kuhn. “Role-Based Access Controls”. In: *Proceedings of the 15th NIST-NCSC National Computer Security Conference*. Baltimore MD, USA: National Institute of Standards and Technology, National Computer Security Center, Oct. 1992, pp. 554–563.

[55] Anna Lisa Ferrara, Georg Fuchsbauer, and Bogdan Warinschi. “Cryptographically Enforced RBAC”. In: *2013 IEEE 26th Computer Security Foundations Symposium*. New Orleans, LA, USA: IEEE, June 2013, pp. 115–129.

[56] Amos Fiat and Moni Naor. “Broadcast Encryption”. In: *Advances in Cryptology - CRYPTO 1993*. Vol. 773. LNCS. Santa Barbara, California, USA: Springer, Aug. 1993, pp. 480–491.

[57] Eiichiro Fujisaki and Tatsuaki Okamoto. “Secure Integration of Asymmetric and Symmetric Encryption Schemes”. In: *Advances in Cryptology - CRYPTO 1999*. Vol. 1666. LNCS. Santa Barbara, California, USA: Springer, Aug. 1999, pp. 537–554.

[58] Juan A. Garay, Jessica Staddon, and Avishai Wool. “Long-Lived Broadcast Encryption”. In: *Advances in Cryptology - CRYPTO 2000*. Vol. 1880. LNCS. Santa Barbara, California, USA: Springer, Aug. 2000, pp. 333–352.

[59] Craig Gentry and Alice Silverberg. “Hierarchical ID-Based Cryptography”. In: *Advances in Cryptology - ASIACRYPT 2002*. Vol. 2501. LNCS. Queenstown, New Zealand: Springer, Dec. 2002, pp. 548–566.

[60] Philippe Golle, Jessica Staddon, and Brent R. Waters. “Secure Conjunctive Keyword Search Over Encrypted Data”. In: *Applied Cryptography and Network Security, Second International Conference, ACNS 2004*. Vol. 3089. LNCS. Yellow Mountain, China: Springer, June 2004, pp. 31–45.

[61] Google. *Google Health*. http://www.google.com/health.

[62] Australian Federal Government. *Personally Controlled Electronic Health Record System (PCEHR) Document*. http://www.yourhealth.gov.au/internet/yourhealth/publishing.nsf/Content/pcehr-document. 2012.

[63] Vipul Goyal et al. “Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data”. In: *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006*. Alexandria, VA, USA: ACM, Oct. 2006, pp. 89–98.

[64] Vipul Goyal et al. “Bounded Ciphertext Policy Attribute Based Encryption”. In: *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*. Vol. 5126. LNCS. Reykjavik, Iceland: Springer, July 2008, pp. 579–591.

[65] Dani Halevy and Adi Shamir. “The LSD Broadcast Encryption Scheme”. In: *Advances in Cryptology - CRYPTO 2002*. Vol. 2442. LNCS. Santa Barbara, California, USA: Springer, Aug. 2002, pp. 47–60.

[66] Lein Harn and Hung-Yu Lin. “A Cryptographic Key Generation Scheme for Multilevel Data Security”. In: *Computers & Security* 9.6 (1990), pp. 539–546.

[67] Michael A. Harrison and Walter L. Ruzzo. “Monotonie Protection Systems”. In: *Foundations of Secure Computations*. Academic Press, 1978.

[68] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. “Protection in Operating Systems”. In: *Communications of the ACM* 19.8 (1976), pp. 461–471.

[69] H. Ragab Hassen et al. “Key Management for Content Access Control in a Hierarchy”. In: *Computer Networks* 51.11 (2007), pp. 3197–3219.

[70] Australian Department of Health and Ageing. *Concept of Operations: Relating to the introduction of a Personally Controlled Electronic Health Record System*. http://www.yourhealth.gov.au/internet/yourhealth/publishing.nsf/Content/CA2578620005CE1DCA2578F800194110/$File/PCEHR-Concept-of-Operations-1-0-5.pdf. 2011.

[71] Florian Hess. “Efficient Identity Based Signature Schemes Based on Pairings”. In: *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002*. Vol. 2595. LNCS. St. John’s, Newfoundland, Canada: Springer, Aug. 2002, pp. 310–324.

[72] Jeremy Horwitz and Ben Lynn. “Toward Hierarchical Identity-Based Encryption”. In: *Advances in Cryptology - EUROCRYPT 2002*. Vol. 2332. LNCS. Amsterdam, The Netherlands: Springer, Apr. 2002, pp. 466–481.

[73] Liang Hu, Zheli Liu, and Xiaochun Cheng. “Efficient Identity-based Broadcast Encryption without Random Oracles”. In: *JCP* 5.3 (2010), pp. 331–336.

[74] Min-Shiang Hwang and Wei-Pang Yang. “Controlling Access in Large Partially Ordered Hierarchies Using Cryptographic Keys”. In: *Journal of Systems and Software* 67.2 (2003), pp. 99–107.

[75] HyperSQL. *HyperSQL DataBase*. http://hsqldb.org/.

[76] Luan Ibraimi et al. “Efficient and Provable Secure Ciphertext-Policy Attribute-Based Encryption Schemes”. In: *Information Security Practice and Experience, 5th International Conference, ISPEC 2009*. Vol. 5451. LNCS. Xi’an, China: Springer, Apr. 2009, pp. 1–12.

[77] InterNational Committee for Information Technology Standards (INCITS). *ANSI INCITS 494-2012 Information Technology - Role Based Access Control - Policy-Enhanced*. http://webstore.ansi.org/RecordDetail.aspx?sku=INCITS+494-2012. Aug. 2012.

[78] Fujitsu Research 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.

[79] JAX-WS. *JAX-WS Reference Implementation*. http://jax-ws.java.net/.

[80] Karthick Jayaraman et al. “ARBAC Policy for a Large Multi-National Bank”. In: *CoRR* abs/1110.2849 (2011).

[81] Audun Josang. “A Logic for Uncertain Probabilities”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 9.3 (2001), pp. 279–212.

[82] Audun Josang. “Artificial Reasoning with Subjective Logic”. In: *Proceedings of the Second Australian Workshop on Commonsense Reasoning*. Perth, Australia, 1997.

[83] Audun Jøsang and Roslan Ismail. “The Beta Reputation System”. In: *Proceedings of the 15th Bled Conference on Electronic Commerce*. 2002.

[84] Audun Josang and Stephane Lo Presti. “Analysing the Relationship between Risk and Trust”. In: *Trust Management, Second International Conference, iTrust 2004*. LNCS. Oxford, UK: Springer, Mar. 2004, pp. 135–145.

[85] Robert J. Jenkins Jr. “ISAAC”. In: *Fast Software Encryption, Third International Workshop, FSE 1996*. Vol. 1039. LNCS. Cambridge, UK: Springer, Feb. 1996, pp. 41–49.

[86] Jonathan Katz, Amit Sahai, and Brent Waters. “Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products”. In: *Advances in Cryptology - EUROCRYPT 2008*. Vol. 4965. LNCS. Istanbul, Turkey: Springer, Apr. 2008, pp. 146–162.

[87] KPMG. *From Hype to Future: KPMG’s 2010 Cloud Computing Survey*. http://www.kpmg.com/ES/es/ActualidadyNovedades/ArticulosyPublicaciones/Documents/2010-Cloud-Computing-Survey.pdf. 2010.

[88] F.H. Kuo et al. “Cryptographic Key Assignment Scheme for Dynamic Access Control in a User Hierarchy”. In: *Computers and Digital Techniques, IEE Proceedings* 146.5 (Sept. 1999), pp. 235–240.

[89] Butler W. Lampson. “Protection”. In: *Fifth Princeton Symposium on Information Science and Systems*. Princeton, NJ, USA, 1971, pp. 437–443.

[90] Butler W. Lampson. “Protection”. In: *Operating Systems Review* 8.1 (1974), pp. 18–24.

[91] Allison B. Lewko and Brent Waters. “Decentralizing Attribute-Based Encryption”. In: *Advances in Cryptology - EUROCRYPT 2011*. Vol. 6632. LNCS. Tallinn, Estonia: Springer, May 2011, pp. 568–588.

[92] Allison B. Lewko et al. “Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption”. In: *Advances in Cryptology - EUROCRYPT 2010*. Vol. 6110. LNCS. French Riviera: Springer, May 2010, pp. 62–91.

[93] Chu-Hsing Lin. “Dynamic Key Management Schemes for Access Control in a Hierarchy”. In: *Computer Communications* 20.15 (1997), pp. 1381–1385.

[94] Chu-Hsing Lin. “Hierarchical Key Assignment without Public-Key Cryptography”. In: *Computers & Security* 20.7 (2001), pp. 612–619.

[95] Huang Lin et al. “Secure Threshold Multi Authority Attribute Based Encryption without a Central Authority”. In: *Progress in Cryptology - INDOCRYPT 2008*. Vol. 5365. LNCS. Kharagpur, India: Springer, Dec. 2008, pp. 426–436.

[96] Ben Lynn. *Pairing-Based Cryptography Library*. http://crypto.stanford.edu/pbc/.

[97] Stephen J. MacKinnon et al. “An Optimal Algorithm for Assigning Cryptographic Keys to Control Access in a Hierarchy”. In: *IEEE Trans. Computers* 34.9 (1985), pp. 797–802.

[98] Behzad Malek and Ali Miri. “Combining Attribute-Based and Access Systems”. In: *Proceedings IEEE CSE’09, 12th IEEE International Conference on Computational Science and Engineering*. Vancouver, BC, Canada: IEEE Computer Society, Aug. 2009, pp. 305–312.

[99] John McLean. “The Algebra of Security”. In: *Proceedings of the 1988 IEEE Symposium on Security and Privacy*. Oakland, California, USA: IEEE Computer Society, Apr. 1988, pp. 2–7.

[100] Peter Mell and Timothy Grance. “The NIST Definition of Cloud Computing (draft)”. In: *NIST special publication* 800.145 (2011), p. 7.

[101] Microsoft. *Microsoft HealthVault*. https://www.healthvault.com/.

[102] Gerome Miklau and Dan Suciu. “Controlling Access to Published Data Using Cryptography”. In: *Proceedings of 29th International Conference on Very Large Data Bases, VLDB 2003*. Berlin, Germany, Sept. 2003, pp. 898–909.

[103] Atsuko Miyaji, Masaki Nakabayashi, and Shunzou Takano. “New Explicit Conditions of Elliptic Curve Traces for FR-reduction”. In: *IEICE Transactions on Fundamentals* E84-A.5 (2001), pp. 1234–1243.

[104] Lik Mui, Mojdeh Mohtashemi, and Ari Halberstadt. “A Computational Model of Trust and Reputation for E-businesses”. In: *HICSS*. 2002, p. 188.

[105] Lik Mui et al. “Ratings in Distributed Systems: A Bayesian Approach”. In: *Workshop on Information Technologies and Systems*. 2001.

[106] OASIS. *eXtensible Access Control Markup Language (XACML)*. http://www.oasis-open.org/committees/xacml/.

[107] OASIS. *Security Assertion Markup Language*. https://www.oasis-open.org/committees/security/.

[108] Sejong Oh and Ravi Sandhu. "A Model for Role Administration Using Organization Structure". In: *7th ACM Symposium on Access Control Models and Technologies, SACMAT 2002*. Monterey, California, USA: ACM, June 2002, pp. 155–162.

[109] Sejong Oh, Ravi Sandhu, and Xinwen Zhang. "An Effective Role Administration Model Using Organization Structure". In: *ACM Transactions on Information and System Security* 9.2 (2006), pp. 113–137.

[110] Rafail Ostrovsky, Amit Sahai, and Brent Waters. "Attribute-Based Encryption with Non-Monotonic Access Structures". In: *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007*. Alexandria, Virginia, USA: ACM, Oct. 2007, pp. 195–203.

[111] Marina Pudovkina. *A known plaintext attack on the ISAAC keystream generator*. Cryptology ePrint Archive, Report 2001/049. http://eprint.iacr.org/. 2001.

[112] Paul Resnick and Richard Zeckhauser. "Trust Among Strangers in Internet Transactions: Empirical Analysis of eBay's Reputation System". In: *The Economics of the Internet and E-Commerce*. Ed. by Michael R. Baye. Vol. 11. Advances in Applied Microeconomics. Elsevier Science. Elsevier Science, 2002, pp. 127–157.

[113] Amit Sahai, Hakan Seyalioglu, and Brent Waters. "Dynamic Credentials and Ciphertext Delegation for Attribute-Based Encryption". In: *Advances in Cryptology - CRYPTO 2012*. Vol. 7417. LNCS. Santa Barbara, CA, USA: Springer, Aug. 2012, pp. 199–217.

[114] Pierangela Samarati and Sabrina De Capitani di Vimercati. "Data Protection in Outsourcing Scenarios: Issues and Directions". In: *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2010*. Beijing, China: ACM, Apr. 2010, pp. 1–14.

[115] Ravi S. Sandh, Venkata Bhamidipat, and Qamar Munawer. "The ARBAC97 Model for Role-Based Administration of Roles". In: *ACM Transactions on Information and System Security* 2.1 (1999), pp. 105–135.

[116] Ravi S. Sandhu. "A Lattice Interpretation Of The Chinese Wall Policy". In: *Proceedings of the 15th NIST-NCSC National Computer Security Conference.* Baltimore, MD, Oct. 1992, pp. 329–339.

[117] Ravi S. Sandhu. "Lattice-Based Access Control Models". In: *IEEE Computer* 26.11 (1993), pp. 9–19.

[118] Ravi S. Sandhu, David F. Ferraiolo, and D. Richard Kuhn. "The NIST Model for Role-Based Access Control: Towards a Unified Standard". In: *ACM Workshop on Role-Based Access Control*. RBAC00. 2000, pp. 47–63.

[119] Ravi S. Sandhu and Qamar Munawer. "The ARBAC99 Model for Administration of Roles". In: *15th Annual Computer Security Applications Conference, ACSAC 1999*. Scottsdale, AZ, USA: IEEE Computer Society, Dec. 1999, pp. 229–238.

[120] Ravi S. Sandhu et al. "Role-Based Access Control Models". In: *IEEE Computer* 29.2 (1996), pp. 38–47.

[121] Andreas Schaad, Jonathan D. Moffett, and Jeremy Jacob. "The Role-Based Access Control System of a European Bank: a Case Study and Discussion". In: *6th ACM Symposium on Access Control Models and Technologies, SACMAT 2001*. Chantilly, Virginia, USA: ACM, May 2001, pp. 3–9.

[122] Jay Schneider et al. "Disseminating Trust Information in Wearable Communities". In: *Personal and Ubiquitous Computing* 4.4 (2000), pp. 245–248.

[123] Adi Shamir. “Identity-Based Cryptosystems and Signature Schemes”. In: *Advances in Cryptology, Proceedings of CRYPTO 1984*. Vol. 196. LNCS. Santa Barbara, California, USA: Springer, Aug. 1984, pp. 47–53.

[124] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*. 2nd. Vol. 106. Graduate Texts in Mathematics. Springer, 2009.

[125] Manachai Toahchoodee et al. “A Trust-Based Access Control Model for Pervasive Computing Applications”. In: *Data and Applications Security XXIII, 23rd Annual IFIP WG 11.3 Working Conference, DBSec 2009*. Vol. 5645. LNCS. Montreal, Canada: Springer, July 2009, pp. 307–314.

[126] Sabrina De Capitani Di Vimercati et al. “Over-encryption: Management of Access Control Evolution on Outsourced Data”. In: *Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB 2007*. University of Vienna, Austria: ACM, Sept. 2007, pp. 123–134.

[127] Sabrina De Capitani di Vimercati et al. “A Data Outsourcing Architecture Combining Cryptography and Access Control”. In: *Proceedings of the 2007 ACM workshop on Computer Security Architecture, CSAW 2007*. Fairfax, VA, USA: ACM, Nov. 2007, pp. 63–69.

[128] Sabrina De Capitani di Vimercati et al. “Encryption Policies for Regulating Access to Outsourced Data”. In: *ACM Transactions on Database Systems* 35.2 (2010).

[129] W3C. *SOAP Message Transmission Optimization Mechanism*. http://www.w3.org/TR/soap12-mtom/.

[130] Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia. “A Logic-Based Framework for Attribute Based Access Control”. In: *Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering, FMSE 2004*. Washington, DC, USA: ACM, Oct. 2004, pp. 45–55.

[131] Brent Waters. "Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization". In: *Public Key Cryptography - PKC 2011*. Vol. 6571. LNCS. Taormina, Italy: Springer, Mar. 2011, pp. 53–70.

[132] *WebMD*. http://www.webmd.com/.

[133] Cungang Yang and Celia Li. "Access Control in a Hierarchy Using One-Way Hash Functions". In: *Computers & Security* 23.8 (2004), pp. 659–664.

[134] Bin Yu and Munindar P. Singh. "A Social Mechanism of Reputation Management in Electronic Communities." In: *CIA-2000 Workshop on Cooperative Information Agents*. Ed. by Matthias Klusch and Larry Kerschberg. Lecture Notes in Computer Science. Springer, 2000, pp. 154–165.

[135] Bin Yu and Munindar P. Singh. "An Evidential Model of Distributed Reputation Management". In: *The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002*. Bologna, Italy: ACM, July 2002, pp. 294–301.

[136] Shucheng Yu et al. "Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing". In: *INFOCOM 2010. 29th IEEE International Conference on Computer Communications*. San Diego, CA, USA: IEEE, Mar. 2010, pp. 534–542.

[137] Eric Yuan and Jin Tong. "Attributed Based Access Control (ABAC) for Web Services". In: *2005 IEEE International Conference on Web Services (ICWS 2005)*. Orlando, FL, USA, July 2005, pp. 561–569.

[138] Fengli Zhang, Qinyi Li, and Hu Xiong. "Efficient Revocable Key-Policy Attribute Based Encryption with Full Security". In: *8th International Conference on Computational Intelligence and Security, CIS 2012*. Guangzhou, China: IEEE, Nov. 2012, pp. 477–481.

[139] Fangming Zhao, Takashi Nishide, and Kouichi Sakurai. "Realizing Fine-Grained and Flexible Access Control to Outsourced Data with Attribute-Based Cryptosystems". In: *Information Security Practice and Experience - 7th International*

Conference, ISPEC 2011. Vol. 6672. LNCS. Guangzhou, China: Springer, May 2011, pp. 83–97.

[140] Sheng Zhong. “A Practical Key Management Scheme for Access Control in a User Hierarchy”. In: *Computers & Security* 21.8 (2002), pp. 750–759.

[141] Lan Zhou, Vijay Varadharajan, and Michael Hitchens. “Achieving Secure Role-Based Access Control on Encrypted Data in Cloud Storage”. In: *IEEE Transactions on Information Forensics and Security* 8.12 (2013), pp. 1947–1960.

[142] Lan Zhou, Vijay Varadharajan, and Michael Hitchens. “Enforcing Role-Based Access Control for Secure Data Storage in the Cloud”. In: *The Computer Journal* 54.13 (Oct. 2011). 0010-4620, pp. 1675–1687.

[143] Lan Zhou, Vijay Varadharajan, and Michael Hitchens. “Generic Constructions for Role-based Encryption”. Technical Report, 2013, submitted to *International Journal of Information Security*, Springer.

[144] Lan Zhou, Vijay Varadharajan, and Michael Hitchens. “Integrating Trust with Cryptographic Role-Based Access Control for Secure Cloud Data Storage”. In: *The 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. Melbourne, Australia: IEEE Computer Society, July 2013, pp. 560–569.

[145] Lan Zhou, Vijay Varadharajan, and Michael Hitchens. “Secure Administration of Cryptographic Role-based Access Control for Large-Scale Cloud Storage Systems”. accepted by special issue in *Journal of Computer and System Sciences* on *Theory and Applications in Parallel and Distributed Computing.*

[146] Lan Zhou, Vijay Varadharajan, and Michael Hitchens. “Trust-based Secure Cloud Data Storage with Cryptographic Role-based Access Control”. In: *Proceedings of the 10th International Conference on Security and Cryptography (SECRYPT)*. Reykjavik, Iceland: SciTePress, July 2013, pp. 62–73.

[147] Lan Zhou, Vijay Varadharajan, and Michael Hitchens. “Trusted Administration of Large-Scale Cryptographic Role-Based Access Control Systems”. In: *The 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom).* Liverpool, United Kingdom: IEEE Computer Society, June 2012, pp. 714–721.

[148] Yan Zhu et al. “Cryptographic Role-based Security Mechanisms Based on Role-Key Hierarchy”. In: *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2010.* Beijing, China: ACM, Apr. 2010, pp. 314–319.

[149] Yan Zhu et al. “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. New York, NY, USA: ACM, May 2013, pp. 33–40.

[150] Yan Zhu et al. “Provably Secure Role-Based Encryption with Revocation Mechanism”. In: *Journal of Computer Science and Technology* 26.4 (2011), pp. 697–710.

[151] Anna Zych, Milan Petkovic, and Willem Jonker. “Efficient Key Management for Cryptographically Enforced Access Control”. In: *Computer Standards & Interfaces* 30.6 (2008), pp. 410–417.