A Secure Role-Based Cloud Storage System For Encrypted Patient-Centric Health Records

Article in The Computer Journal · July 2016
DOI: 10.1093/comjnl/bxw019

CITATIONS 0
READS 22

3 authors, including:

Vijay Varadharajan
Macquarie University
414 PUBLICATIONS 3,040 CITATIONS

Kanchi Gopinath
Indian Institute of Science
84 PUBLICATIONS 1,001 CITATIONS

Some of the authors of this publication are also working on these related projects:

Access Control and Identity in IoT View project

All content following this page was uploaded by Vijay Varadharajan on 04 January 2017.
The user has requested enhancement of the downloaded file. All in-text references underlined in blue are added to the original document and are linked to publications on ResearchGate, letting you access and read them immediately.
A Secure Role-Based Cloud Storage System For Encrypted Patient-Centric Health Records

LAN ZHOU1, VIJAY VARADHARAJAN1*, AND K. GOPINATH2

1Advanced Cyber Security Research Centre, Macquarie University, Sydney, NSW 2109, Australia
2Department of Computer Science and Automation, Indian Institute of Science, Mathikere, Bengaluru, Karnataka 560012, India
*Corresponding author: vijay.varadharajan@mq.edu.au

With the rapid developments occurring in cloud services, there has been a growing trend to use cloud for large-scale data storage. Due to the increasing popularity of cloud storage, many healthcare organizations have started moving electronic health records (EHRs) to cloud-based storage systems. However, this has raised the important security issue of how to protect and prevent unauthorized access to EHR data stored in a public cloud. Several cryptographic access control schemes have been proposed to protect the security of data stored in the cloud by integrating cryptographic techniques with access control models. In this paper, we consider a novel role-based encryption technique to build a secure and flexible large-scale EHR system where role-based access control policies are enforced in a cloud environment. Then we discuss a practical EHR system called the personally controlled electronic health record (PCEHR) system recently developed by the Australian Government, and show how the security weaknesses in the PCEHR system can be addressed by our proposed scheme. The proposed system has the potential to be useful in commercial healthcare systems 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.

Keywords: secure encrypted health records; role-based encryption; cloud data storage security; secure role-based access control

Received 23 March 2015; revised 15 January 2016
Handling editor: Andrew Martin

1. INTRODUCTION

There has been a growing trend in the recent times to store data in a cloud due to the dramatic increase in the amount of digital information such as consumers’ personal data and enterprise data for back up or archiving. A recent survey [1] shows that 69% of enterprises have either applications or infrastructure running in the cloud today, compared to 57% in 2012, and cloud investments have also increased by 19%. There are different types of clouds such as public cloud, private cloud and hybrid cloud [2]. A public cloud is often defined as a multi-tenant environment, where one can buy certain portion of resources from the infrastructure, which is shared with a number of other clients or tenants. A private cloud on the other hand is a single-tenant environment where the hardware, storage and network are dedicated to a single client or company. Whether a public cloud or private cloud, the key benefits of cloud computing include scalability of computational and storage resources, instant provisioning and virtualized resources. In this paper, when we use the word cloud, we are referring to a public cloud. Cloud data storage can be particularly attractive for users, individuals or enterprises, with unpredictable storage demands, requiring an inexpensive storage tier or a low-cost, long-term archive. By outsourcing users’ data to a public cloud, service providers can focus more on the design of functions to improve user experience of their services without worrying about resources to store the growing amount of data. Cloud can also provide on demand resources for storage, which can help service providers to reduce their maintenance costs [3]. Furthermore, storage in a public cloud can provide a flexible and convenient way for users to access their data from anywhere on any device.
In recent years, use of public clouds to store personal health records (PHR) online has become more popular. A number of cloud providers have started providing services, which allow patients’ health data to be used more easily, such as the Microsoft HealthVault [4], Google Health [5] and WebMD [6]. The number of electronic health records (EHRs) is expected to grow even larger in the coming years as more facilities adopt electronic records, and rely increasingly on mobile applications and devices such as tablets and smartphones to gather this patient information [7].1 In Australia, the Government has recently announced an EHR system called the personally controlled electronic health record (PCEHR) system [8] to assist patients in better organizing their PHR and provide the patients with flexibility in controlling the access to their PHR.

A typical way to manage patients’ PHR data that is adopted by most of the healthcare providers is to store them on local storage servers. Since each healthcare provider keeps the PHR individually, it is usually inconvenient for patients as their medical records cannot be shared efficiently among healthcare providers. 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/he will take the blood test results by hand or will ask for an electronic copy of result to be sent via email in advance or will need to do the blood test again. 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 have been accessed by unauthorized persons. Using a public cloud2 to store PHRs could make data sharing among different healthcare providers and patients easier. Furthermore, there could also be other benefits of using a cloud to store data, as it can be used to provide additional services such as notifying the patients as to who has accessed their records and when.

In order to provide a fast and convenient access to health information for individuals, Australian Government launched a 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 organizations. 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.

When patients consider storing their PHR in the cloud, security is often their main concern. In the PCEHR system, users need to trust the system and the operators to protect the security and privacy of their sensitive health information. However, since PCEHR data are 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 in an unauthorized manner, without users’ permissions. In addition, flaws can also arise in the 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 unauthorized healthcare organizations as the data are stored in plain form in the PCEHR system.

In this paper, we present the design of a secure EHR system using role-based encryption (RBE) schemes. In our system, PHR data are encrypted before being stored in the system, and can only be decrypted by users who are allowed by the access policies. Unlike in the PCEHR system, if the PHR data are accessed by unauthorized parties, such as employees of the cloud provider, it will not cause the leak of patients’ sensitive health information. Additionally, compared to the PCEHR system, it allows users to have more flexible control over their PHR data stored in the system than the existing access control mechanisms adopted by PCEHR. Moreover, it can address several security issues such as secure transfer of privileges between of healthcare providers in the PCEHR system. We describe the architecture of our system and show that it can work either as a standalone EHR storage system or a security enhancement component that can be integrated with other EHR systems such as PCEHR. Then, we illustrate using an example how our system can provide a practical solution for securely storing PHR data.

The rest of this paper is organized as follows. Section 2 gives an overview of the PCEHR system and reviews the definition of the RBE schemes. In Section 3, we present the design of our EHR system, and discuss the system architectures. The access control mechanisms used in the PCEHR system are discussed in Section 4. In Section 5, we use examples to illustrate how our designed system operates and overcomes the security challenges and issues in the PCEHR system. Finally, Section 6 concludes the paper.

2. BACKGROUND

In this section, we first present an overview of the PCEHR system introduced by the Australian Government, which is a target application for our proposed system in this paper. Then, we describe the RBE schemes that provide us the tools to build a secure EHRs storage system.
2.1. **Australian PCEHR system overview**

The PCEHR system [8] was introduced by the Australian Government in 2012 to help patients share efficiently their health information. First, we outline an example scenario of showing the various parties in the PCEHR system in Fig. 1. Health information about the patients are stored in the PCEHR system; various parties such as the patients, general practitioners (GPs) and specialists may wish to access the health data of the patients from the PCEHR system for different purposes. Initially, a patient registers for a PHR. Then, the clinical providers upload information into the record. For instance, when a patient goes to see a GP or a specialist, she or he may wish that the GP or the specialist access her or his PHR data directly from the PCEHR system. In the case of an emergency, there needs to be a emergency access policy, which will allow a doctor from the emergency department to access the patient’s PHR data directly. Meanwhile, the administration in 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 a system, and healthcare providers and organizations who wish to access the stored health information 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 either via the Provider Portal interface of the PCEHR system or via the local clinical systems built and owned by themselves. The healthcare providers can seek permissions to access users’ PCEHR data, to search and view the PCEHR data of individual users and to upload clinical documents.

In the PCEHR system, the PCEHR data are a collection of health documents stored in a network of connected registered repositories. A multilayer approach that consists of firewalls, gateways and portals is used to ensure only authorized users can access the PCEHR system; that is, the security of the system is protected by firewalls and gateways, which are set up between the public users and the storage system. These devices are used in the provision of authentication and authorization security services. Individual users/patients make use of username/password mechanism to authenticate themselves and log into the PCEHR system. An authorized user who wishes to access the PCEHR data on behalf of a healthcare organization first needs to be authorized by the organization. When an authorized user wishes to access the PCEHR system from the clinical system of the organization, the local clinical system needs to authenticate itself to the PCEHR system using the organization’s digital credentials and pass on the user details.

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 responsible to 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, loss, unauthorized access, unauthorized 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 authorized 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.

![FIGURE 1. PCEHR system example scenario.](image-url)
trusted to prevent unauthorized users from accessing their data. Role-based access control (RBAC) is a well-known access control model, which provides flexible controls and security management by having two mappings, users to roles and roles to privileges on data objects. In this paper, we describe the use of a secure RBAC-based cloud storage system where the access control policies are enforced by a new RBE that we proposed earlier in Refs [9, 10]. In our RBE scheme, the owner of the data encrypts the data in such a way that only the users with appropriate roles as specified by a RBAC policy can decrypt and view the data. For instance, if the cloud provider does not have an appropriate role, then the cloud provider will not be able to access and view the plain data, thereby reducing the trust requirement on the cloud provider.

The RBAC model was formally introduced in 1992 [11]. 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 [12], and the RBAC standard was proposed in 2000 [13]. In Ref. [13], four different types of RBAC have been defined, 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. In this paper, when we use the term RBAC, we are referring to a hierarchical RBAC system. With RBAC, access decisions are based on the roles that individual users have been assigned to. 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. It simplifies the administration and management of permissions; roles can be updated without updating the permissions for every user on an individual basis.

A cryptographic RBAC scheme integrates an encryption scheme with the RBAC model to enforce the access control policies in an outsourcing 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 in an untrusted cloud environment and the stored ciphertext can only be decrypted by those, which are allowed by the access policies. A hierarchical cryptographic access control scheme [14] was proposed 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. The problem of access control for securely outsourcing data using cryptographic techniques was first considered in Ref. [15]. An improved scheme was proposed in Ref. [16] to address access policy updates. Several cryptographic access control approaches have been investigated in Refs [9, 17] to address the problem of secure data access and cost effective key management in distributed environments. Subsequently, a two-layer encryption model was proposed in Ref. [10] 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.

In Refs [18, 19], new RBE schemes have been proposed. In these schemes, the ciphertext and the decryption key that the user needs to keep are constant in size, and the user memberships are managed by individual roles as opposed to a central administrator like in other cryptographic RBAC schemes. Since RBE schemes have superior characteristics such as constant size ciphertext and decryption key compared to other cryptographic RBAC schemes, we have chosen RBE schemes to build our secure EHR system.

First, we review the four types of entities in the RBE scheme. SA is a system administrator that has the authority to generate the keys for users and roles, and to define the role hierarchy. RM is a role manager who manages the user membership of a role. Owners are the parties who want to store their data securely in the cloud. Users are the parties who want to access and decrypt the stored data in the cloud. The data are stored in a public cloud and the public cloud infrastructure provides interfaces so that all the other entities (such as the SA, RM, users and owners) can interact with the stored data in the public cloud. Next, we review the algorithms of the RBE scheme:

\[ \text{Setup} (\lambda) \text{ takes as input the security parameter } \lambda \text{ and outputs a master secret key } mk \text{ and a system public key } pk. \ mk \text{ is kept secret by the SA while } pk \text{ is made public to all users of the system.} \]

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

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

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

\[ \text{RevokeUser} (pk, sk_R, U_R, ID_U) \text{ is executed by a role manager RM of a role } R \text{ to revoke the role membership from a user } ID_U, \text{ which also results in the role public parameters } pub_R \text{ and role user list } U_R, \text{ being updated.} \]
Encrypt \( (pk, R, M) \) 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 \( R \), and outputs the ciphertext \( C \).

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 \( M \).

Besides RBAC, there are also other access control models such as attribute-based access control (ABAC). In ABAC, access is granted based on attributes of the user. Systems define combination of attributes as the access policies, and users need to prove that they have these attributes in order to gain access. In 2006, the first attribute-based encryption (ABE) scheme was proposed in Ref. [20] based on the work in Ref. [21], and some other ABE schemes have been proposed afterwards. In these schemes, data are encrypted to a set of attributes, and users who have the private keys associated with these attributes can decrypt the data. These works have provided an alternative approach to secure the data stored in a distributed environment using a different access control mechanism. We have shown [9] how the ABE scheme can be modified to enforce RBAC policies in Ref. [9]. However, with the ABE, the size of user key is not constant, and furthermore, user revocation poses significant issues as it results in the key update of all the other users of the same role. This is a significant disadvantage.

### 2.3. RBE scheme construction

In this section, we propose an RBE scheme, which is designed using asymmetric bilinear groups described in Subsection 2.2.1. This RBE scheme is used in our cloud storage system to enforce the RBAC policies.

Now we describe the RBE scheme as follows:

**Setup:** Generate three groups \( G_1, G_2, G_T \), and a bilinear map \( \hat{e} : G_1 \times G_2 \rightarrow G_T \). Randomly choose two generators \( g \in G_1 \) and \( h \in G_2 \), two secret values \( s, k \leftarrow \mathbb{Z}_p \) and two hash functions \( H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p \), \( H_2 : G_T \rightarrow G_1 \). The master secret key \( mk \) and system public key \( pk \) are defined as

\[
mk = (s, k, h), \quad pk = (w, v, g^k, g^g, \ldots, g^{g^q})
\]

where \( w = h^t \), \( v = \hat{e}(g, h) \) 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 \( ID = ID_U \) is an identity of a user \( U \), SA computes the user secret as

\[
dk_U = h^{s+H_1(ID_U)}
\]

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 \( ID = ID_R \) is an identity of a role \( R \), SA first computes the role secret as

\[
sk_R = g^{s+H_1(ID_R)}
\]

and gives \( sk_R \) to the role manager \( R \) together with the \( RULC_R \), which is initially set to empty.

**ManageRole(mk, ID_R, \( \mathcal{PR}_R \)):** Assume that \( \mathcal{PR}_R \) is a set of identities \( \{ID_{R_1}, \ldots, ID_{R_n}\} \) 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 \((A_R, B_R, RULC_R)\) as role public parameters in the cloud where

\[
A_R = h^{s+H_1(ID_R)} \prod_{i=1}^n (s+H_1(ID_{R_i})), \quad B_R = A_R^k
\]

**AddUser(pk, sk_R, RULC_R, ID_U):** Assume that the role manager \( RM \) of role \( R \) wants to add a user \( U_i \) with the identity \( ID_{U_i} \) to the role. \( RULC_R \) is the set of \( n \) users who belong to the role \( R \) and \( U_i \) is not in \( RULC_R \). The role manager RM first sends the identity \( ID_{U_i} \) to the cloud. When receiving the user’s identity, the cloud computes the value

\[
Y_i = g^{s+H_1(ID_{U_i})} \prod_{i=1}^n (s+H_1(ID_{U_i}))
\]

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:

\[
\hat{e}(Y_i', w \cdot h^{H_1(ID_{U_i})}) = \hat{e}(Y_i, h)
\]
If the equation holds, RM chooses two random values \( r_i, t_i \) \( \sim \mathbb{Z}_p \), if \( Y_i \) is received from cloud for the first time, or uses the existing \( r_i, t_i \) otherwise. Then, it computes
\[
K_i = v^{r_i}, \quad T_i = g^{-t_i}, \quad W_i = w^{-r_i},
\]
\[
V_i = Y_i^{r_i} = g^{r_i (s+H_1(ID_{uk}))} \prod_{j=1,j\neq i}^m (s+H_1(ID_{uk}))^{-1}.
\]
\[
S_i = H_2(K_i) \cdot \sk_R \cdot g^{k_i} = H_2(v^{r_i}) \cdot g^{s+H_1(ID_{uk})+H_i}.
\]
RM adds \( ID_{uk} \) into \( \mathcal{RLUL}_R \), 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
\[
(ID_R, W_i, V_i, S_i, \mathcal{RLUL}_R).
\]

**RevokedUser(pk, sk_R, \mathcal{RLUL}_R, ID_{uk}):** To revoke a user \( U_k \) from a role \( R_i \) that has a set \( N \) of \( n \) users in \( \mathcal{RLUL}_R \), and \( U_k \in N \). The role manager RM first removes \( ID_{uk} \) from role user list \( \mathcal{RLUL}_R \) and sends the user identity \( ID_{uk} \) to the cloud. When receiving the user’s identity, the cloud computes the value
\[
Y_i = g^{\prod_{j=1,j\neq i}^m (s+H_1(ID_{uk}))}
\]
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:
\[
\hat{e}(Y_i, h) = \hat{e}(Y_i, w \cdot H_1(ID_{uk})).
\]
If the equation holds, RM chooses two random values \( r_i', t_i' \) \( \sim \mathbb{Z}_p \) and recomputes
\[
K_i = v^{r_i'}, \quad T_i = g^{-t_i'}, \quad W_i = w^{-r_i'},
\]
\[
V_i = Y_i^{r_i'} = g^{r_i' \prod_{j=1,j\neq i}^m (s+H_1(ID_{uk}))},
\]
\[
S_i = H_2(K_i) \cdot sk_R \cdot g^{k_i'} = H_2(v^{r_i'}) \cdot g^{s+H_1(ID_{uk})+H_i}.
\]
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):** Assume that the owner of the data \( M \) wants to encrypt \( M \) for the role \( R_i \). The owner randomly picks \( z \) \( \sim \mathbb{Z}_p \), and computes
\[
C_1 = w^{-z}, \quad C_2 = A_{R_i} z, \quad C_3 = B_{R_i} z, \quad K = v^z.
\]
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, dk_{uk}, C):** Assume that a role \( R_i \) has a set \( \mathcal{R} \) of ancestor roles, and the set \( \mathcal{M} = \mathcal{R}_i \cup \mathcal{R} \) has \( m \) roles \( \{R_1, \ldots, R_m \} \). \( R_i \in \mathcal{M} \) is one ancestor role of \( R_i \), and there is a set \( N' \) of \( n \) users \( \{U_1, \ldots, U_n \} \) 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
\[
D = \hat{e}(T_i, C_3), \quad g^{R_i(x)}(s), \quad g^{R_i(x)}(z),
\]
\[
Aux_1 = \prod_{j=1,j\neq i}^m H_i(ID_{uk}), \quad Aux_2 = \prod_{j=1,j\neq k}^m H_i(ID_{uk})
\]
where
\[
p_{k,M}(s) = \frac{1}{s} \cdot \left( \prod_{j=1,j\neq i}^m (s + H_1(ID_{uk})) - \prod_{j=1,j\neq i}^m (H_1(ID_{uk})) \right)
\]
\[
p_{k,N}(s) = \frac{1}{s} \cdot \left( \prod_{j=1,j\neq k}^m (s + H_1(ID_{uk})) - \prod_{j=1,j\neq k}^m (H_1(ID_{uk})) \right)
\]
Then, the cloud returns the following tuple and the ciphertext of \( M \) to the user \( U_k \),
\[
(C_1, C_2, D, g^{R_i(x)}(s), g^{R_i(x)}(z), Aux_1, Aux_2)
\]
After receiving the ciphertext from the cloud, \( U_k \) recovers the data \( M \) by computing
\[
K = (\hat{e}(g^{R_i(x)}(s), C_1) \cdot \hat{e}(S_i \cdot H_2(K_i)^{-1}, C_2) \cdot D)^{-1}_{\text{Aux}_1}
\]
where
\[
K_i = (\hat{e}(V_i, dk_{uk}) \cdot \hat{e}(g^{R_i(x)}(s), W_i))^{-1}_{\text{Aux}_2}
\]
By using the key \( K \), \( U_k \) can decrypt the ciphertext of \( M \) and recover the data \( M \).

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

**Security Analysis:** We have shown that our scheme is semantically secure under the General Decisional Diffie-Hellman
Exponent (GDDHE) assumption introduced in Ref. [23] by defining a specific GDDHE problem. Security properties and their proofs are provided in the PhD Thesis [24].

2.4. Example scenario

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

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

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

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

2.5. Extended RBE with 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 authorized users can decrypt the data. In this subsection, we show an extension of our proposed RBE scheme, which supports the encryption for multiple roles. In the extended scheme, only the $Encrypt$ and $Decrypt$ algorithms are modified from the ones in the original scheme, and the other algorithms are the same as in the original RBE scheme. The two modified algorithms in the extended RBE scheme are described as follows:

$Encrypt(pk, \{p_{R_i}\}_{x \in [1,l]}):$ 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

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

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

$Decrypt(pk, pub_{R_i}, dk_{U_k}, C):$ Assume that each role $R_i$, $x \in [1,l]$ in the set to which the data are encrypted has an ancestor role set $\mathcal{R}_i$, and a role $R_i \in \mathcal{R}_i$ is an ancestor role of $R_j$ where $1 \leq j \leq l$. We denote $\mathcal{N}$ as the $n$ user members $\{U_1, \ldots, U_n\}$ of $R_i$, and $\mathcal{M} = R_i \cup \mathcal{R}_i$. 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 $M$ to the user $U_k$ in the same way as in the original RBE scheme,

$$(C_1, C_2, D, g^{n_{\mathcal{M}(x)}}, g^{n_{\mathcal{N}(x)}}, Aux_1, Aux_2)$$

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

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

In order to provide a solution for securely storing PHRs in a public cloud, we have developed a EHR system using RBE schemes. The system protects the stored PHR data’s confidentiality and prevents unauthorized access to the PHR data stored in the system while allowing patients to have flexible control over their PHR data. In this section, we present the design of our secure EHR storage system. We first describe the design of the PHR structure. Then, we discuss two different architecture scenarios in which our system can be used.

3.1. Patient-centric PHR structure

In general, the patient record structure should reflect the way that patients and clinicians can work together to achieve high-quality care for the patients. Typically, a patient record is organized into different categories or compartments, sometimes comprising fixed data formats and sometimes as free text. Although there have been several different formats proposed over the years [25], there have also been standardization efforts of EHRs [26]. For the purposes of our paper, to allow flexible control over the PHR data, we assume that a patient record data consist of different categories of information relevant to different healthcare professionals. For instance, Fig. 3 shows an example where the PHR data of each patient are divided into several different categories organized in a hierarchical structure. The actual categories are given as an example for the purpose of illustration. In later discussions of our proposed system that supports role-based access policies, each category in this example represents a role.

The specific structure of a patient record is not the focus of this paper. Instead our aim is to show that how our system can work with a generic patient health record structure in which there are different categories and there are specific hierarchical relationships between these different categories. In practice, it is important that the structure should be able to capture accurately key data relating to the patient details and patient’s medical conditions, core transactions of healthcare such as the record of the patient/clinician dialogue, diagnosis and the resulting decisions and actions taken, and they must be accessible to patients and clinicians in an appropriate manner for the care to be delivered. This task can be simplified by defining a common role hierarchy as a template for all the newly joining patients. All patients can start with the same role hierarchy, which is intended to capture the most medical cases; then, the patients can modify their role hierarchies depending on the actual needs.

In principle, there must be provisions for extensibility of the structure to take into account the specific circumstances of the patients. For example, it should be possible to extend the list of categories of the specialists 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 organizations can also be extended depending on those with whom the patient wishes to share the PHR data. An external organization may have its own hierarchical structure. Therefore, when a patient shares his/her PHR data with an external organization, it will be the external organization that decides which individual user can access the data. In this example, we only consider such an organization 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.

In the illustrated 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 immunization history, medications and allergies. General Practitioner stores the medical records that 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/his PHR data.

Let us now consider multiple access levels on the PHR data using such a structure. Consider, for instance, the read permission inheritance. For example, the receptionist of the cardiologist can only read the personal details of the patient (the arrow from Reception to Personal Details indicates read permission inheritance), 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. (This inheritance indicated by the arrow from Cardiology to General Practitioner to Basic Medical Records.) In general, restrictions on the write permission may not be required when storing PHR data in a cloud system; that is, any healthcare provider (any role) can be allowed to write notes to a patient’s record. However, in some cases, a patient may want only certain authorized users to write data onto his/her PHR. If this is the case, then the write permission can be assigned independently of the read permission. For example, the patient can encrypt the role public parameters for encryption to the role Personal Details, and hence only users who have been assigned a role by the patient can write PHR data for that patient.

In the case of PCEHR system, the PCEHR operation [27] specifies the health information is categorized into several different clinical document types. We use these document types defined in the PCEHR specification in our following discussion. Now we describe what these document types are and how they can be associated to the categories in the designed PHR structure. Using these document types in the following description will help to make the comparison of our system and the PCEHR easier.

**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 immunizations. 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 GP. This type of document contains only the basic health information of a user. Hence, it should be accessible to all the specialist and GP roles in the system for diagnosis and treatment. Patients can decide which information they wish to share. 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 regarding the discharge of a user from a hospital. This type of document should be encrypted to the specialist role who is in charge of the treatment of the patient 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 organization for the purpose of referring the user to the receiving organization. Usually, the referrals are created by a GP to refer a patient 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

---

3Our aim is to consider options to enhance the PCEHR or consider a solution that can work as a standalone system.
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.

3.2. System architecture

Next, we look at the architecture of our PHR storage system. As mentioned previously, our system can work as either a standalone EHR storage system or a security enhancement component that can be integrated with other EHR storage systems such as the PCEHR system. Our system is implemented using the RBE schemes described in both Refs [18] and [19]. 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 is shown in Fig. 4.

3.2.1. Standalone PHR storage system

The PHR storage system can be deployed in the cloud for providing services to public users as shown in Fig. 4a. In this scenario, we adopt the architecture presented in Ref. [19], which is a hybrid of a private cloud and a public cloud. The private cloud is set up and managed by the organization, 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 predefined 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/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/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 belong to and the access policies are stored in a private cloud to which only authorized 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. (That is, the decryption keys are generated at the client and this is done in an automatic manner; they are not generated or computed in the cloud. The cloud is only used to compute auxiliary parameters from publicly available role parameters). When a doctor wants to upload a clinical document for a user, she/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. Once again in the implementation, these are done in an automatic manner. 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 are encrypted before being stored in the cloud, the privacy will be guaranteed even if unauthorized users have accessed the stored data in the cloud.

3.2.2. EHR systems’ security enhancement component

Our PHR storage system can also work with general EHR systems as a security enhancement component used by individual healthcare organizations. Figure 4b shows the architecture when a healthcare organization uses our PHR storage system to access the PCEHR system. In this scenario, we use the PCEHR system as an example 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 uses the RBE scheme implementation described in Ref. [18], 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, 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, use of this RBE scheme does not affect the performance of the system in terms of user management.

Note that most of the EHR systems including PCEHR do not provide any computing service, and 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 organizations to access the system via their own local clinical systems. Our PHR storage system can therefore be deployed in local clinical systems of healthcare organizations 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 the secret key. When a doctor of a healthcare organization 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.

3.2.3. Experimental results

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 [28] web services, and the web services are hosted in Apache Tomcat. The clouds use HyperSQL [29] database, which can be easily replaced by other databases for server side data storage. The client side is written as Java Applet, which can run in any Internet browser with Java support. To ensure that the client side gets the valid system public keys, these keys are embedded in the Java Applet, and the Applets are signed by the key generated by the trusted certificate authority when the Java Applet is compiled.

Our RBE scheme uses asymmetric bilinear groups, where the bilinear map takes inputs from two distinct isomorphic groups \( \mathbb{G}_1, \mathbb{G}_2 \). This allows a greater variety of pairings to be used in the implementation, especially certain families of non-supersingular elliptic curves [30]. In our implementation, we use a 163-bit MNT curves [31] with the embedding degree of 6. In practice, the security of a 160-bit elliptic curve is roughly equivalent to 1024-bit RSA [32]. 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 [33] as the symmetric encryption algorithm \( Enc \); the reason for this choice is that a stream cipher can work with the W3C message transmission optimization mechanism [34] 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 \( 4.67 \times 10^{1240} \) [35] for the best known attack to ISAAC [36].

We use jPBC [37] and PBC [38] as our pairing-based cryptography library (jPBC wraps the PBC library to generate a MNT curve), and Bouncy Castle crypto library [39] for SHA-1 and ISAAC.

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.

\[ \text{If a user loses or leaks the secret key, the user can request the PHR storage system to block the access to her/his PHR data. Then, the user can generate a new secret key, use the emergency access discussed in Section 5 to recover all her/his PHR data in the system and re-encrypt them.} \]
Let us first consider the size of the ciphertext. In our RBE scheme, 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. So an important characteristic of our scheme is 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. Table 1 shows the sizes of the ciphertext for plaintext sizes of 1000 bytes, 10000 bytes and 100000 bytes when the target role has 10, 100 and 1000 ancestor roles, respectively. First, we note that the differences in size between the plaintext and ciphertext are constant. Second, the ciphertext size remains the same when the number of ancestor roles changes.

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

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

Figure 5 shows the time that the cloud server has spent in executing the decryption algorithm on a ciphertext of a 1 KB file when different number of users are in the role to which the user performing the decryption belongs. In this case, four 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 cloud environment having 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 previous computation. So it does not have to compute for every decryption request, which reduces the average cloud decryption time even further.

### Table 1. Ciphertext size (bytes).

<table>
<thead>
<tr>
<th>Plaintext size</th>
<th>10 Roles</th>
<th>100 Roles</th>
<th>1000 Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1432</td>
<td>1432</td>
<td>1432</td>
</tr>
<tr>
<td>10000</td>
<td>10432</td>
<td>10432</td>
<td>10432</td>
</tr>
<tr>
<td>100000</td>
<td>100432</td>
<td>100432</td>
<td>100432</td>
</tr>
</tbody>
</table>

![Graph](http://example.com/graph.png)  
**Figure 5.** Public cloud operation time. (a) Public cloud decryption time and (b) public cloud operation time in user management.
In our system, role managers also outsource part of the computations to the cloud, which are concerned with user management. Figure 5b shows the time that the cloud has spent in collaborating with role managers on this computation; in this experiment, we have created four 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 6a shows the time for encrypting and decrypting files of different sizes on the client side. In this experiment, we created 5 roles and 10 users in each role. In our measurements, the encryption time was measured from the time when an owner clicks on the upload button in the Java Applet after choosing the file to be encrypted, to the time when the file upload has been completed and the owner receives the cloud’s response indicating that the transaction successful. The decryption time was measured from the time when a user starts receiving the ciphertext from the cloud till the time the plaintext is saved to a file on the local disk drive.

Since we are using the stream cipher ISAAC in our implementation, the encryption/decryption of data can happen while the data are being transferred over the network. In the encryption/decryption algorithm of the RBE scheme, a key is computed and used in the symmetric encryption scheme. As seen from Fig. 6a, when the size of the plaintext is smaller than 100 KB, 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 1 MB, 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 of auxiliary parameters to the cloud, the operations’ time of clients is reduced dramatically. Figure 6b shows the time that we measured for the encryption of 1 KB data by the owner, decryption of 1 KB data by the users and the user management of role managers (when a role contains different number of users). Once again there are four 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.

These experimental results show that the proposed scheme and architecture have several practical advantages such as fixed ciphertext size and the users having to keep a single key for decryption as well as system operations being efficient regardless of the complexity of the role hierarchy and user membership in the system.

4. PCEHR SYSTEM

In the PCEHR system, the privacy of the stored EHR is protected by certain access control policies, which can be chosen by the users. In this section, we first review the access control mechanisms that are used by the PCEHR system. Then, we discuss several challenges and issues associated with using these access control methods.

4.1. 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 look at the access control mechanisms that are currently used by the PCEHR system.

4.1.1. 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 organizations to or from the access list. Only the organizations 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 organizations on the access list will be removed automatically if they have not accessed any of the user’s PCEHR data for 3 years or more, and the user needs to grant the permission to the organizations again if she or he wishes to allow the organizations to access the PCEHR data. Note that the access control policy is enforced by the system itself in the PCEHR system.

4.1.2. 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 organization involved in the care of a user is added to the user’s access list automatically. 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 organization. The ‘removed’ clinical document can be later restored by the system operator upon the request of the user.

4.1.3. 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 organizations 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 organization on the access list as being ‘revoked’ if the user does not want the organization to access her or his PCEHR data.

Provider access consent code. Under the advanced access control settings, users are able to set up provider access consent code (PACC), which is used as a PIN or passphrase. Organizations will not be able to add themselves to the access list unless they have a valid PACC. The organizations, which have already been added on the access list, can access the users’ PCEHR with a valid PACC. To strengthen privacy and security, the ability of healthcare provider organizations to override the requirement for a PACC (where a consumer has forgotten his or her PACC) has been removed. If the user has set up advanced access controls that require a PACC to access their PCEHR, then the policy requires healthcare providers to receive that code and enter it to access the consumer’s PCEHR for the first time. If a user has forgotten or lost her/his PACC, the user will have to contact the PCEHR System Operator to either retrieve or reset the 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 organization. The available levels are as follows:

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

2. Limited access: The clinical documents at this level are accessible only to a more limited group of healthcare organizations as specified by users. The healthcare organization 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 organization so as to ensure only the authorized 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 organization has when it uploads the document. In some cases, a user may want to set different access levels to the documents from the same organization. In that case, the user can change the access level accordingly after the documents have been uploaded to the system.

By default, healthcare organizations 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 organization 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.

4.1.4. 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, and the registered healthcare organizations believe that this is necessary to lessen or
prevent a serious threat to the user or another individual’s life, health or safety. Emergency access is a requirement under the Australian Privacy Principles of the Privacy Act. An example of such an emergency situation arises if the user is unconscious for some reason. Another example could be where a dangerous infection has been detected within a hospital and it is necessary to identify the source of the infection to prevent its spread. If such conditions do not apply when emergency access is gained, the healthcare provider will be breaching the law and penalties will apply.

Emergency access will override the access control policies that are set on users’ PCEHR data. However, it is not required for the users who are using basic level access controls, as the organization, which is providing healthcare services to the users will be granted 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 organizations, 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 organization to the users’ access list if it is not already on the list and provide the organization with access to both ‘limited access’ and ‘general access’ clinical documents. The organization’s access level will be reverted to the previous access level prior to emergency access after a period of 5 days from the time of last access to the user’s PCEHR data. If the access is still required after the timeout, the healthcare organization could assert emergency access again. There is no limit on how many times the emergency access can be used, though the organization may want to obtain a more persistent form of access from the user or her/his authorized representatives.

To prevent the abuse of emergency access, all uses of emergency access are audited and recorded by the PCEHR system. The users can view the audit logs. Furthermore, emergency accesses to PHRs can also be notified via SMS or email from the System Operator.

4.1.5. Forward consent
When a user is referred by a healthcare organization to another organization, it may be necessary for the referred organization 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.

4.2. 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 is stored in a network of connected systems exposing a number of system interfaces enabling different services and applications to connect to the PCEHR. Security risks can arise as users and services in such a distributed system may access the PCEHR data without proper security authorizations. This can arise due to several reasons, from security system misconfigurations to flaws arising in system implementations to inconsistencies arising in authorization policies to errors in the enforcement of authorization policies specified on the PCEHR data. Our main issue is that any of these flaws in the system software could potentially lead to leakage of patients’ sensitive health information to unauthorized users and healthcare organizations as the data are stored in the plain form in the PCEHR system.

Now we will use the example shown in Fig. 1 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 organizations with special permits can access. For example, a patient may wish that her/his GP can access only documents with ‘general access’ by giving general access to the GP, and limited access to the specialists. Now let us assume that two specialists, namely a cardiologist and an endocrinologist, are involved in providing care to the 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: Now let us assume that a user has moved to a different suburb, and wishes to change to a different GP. In the PCEHR system, a healthcare organization 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 organizations 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 organization, which knows the PACC or PACCX.
Forward Consent: When a user is referred by a healthcare organization to another organization in the PCEHR system, the referred organization needs to contact the user to obtain the access to the user’s PCEHR data. This could be an issue if the referred organization 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. Creating de-identified data poses several challenges and it may not be possible to prevent leakage in all cases. For example, a patient who has had a special disease can be identified easily from her/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.

5. PHR SYSTEM SECURE OPERATIONS

Now consider a general healthcare information storage system, which uses the cloud to store patients’ PHR data. 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 organizations.

In this section, we show that our designed system addresses the weaknesses of the PCEHR system and meets the above-described security requirements.

5.1. System operations

Now, we describe the operations in our PHR storage system to show how the weaknesses in the PCEHR system can be remedied in our 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 have been written to the cloud have the signature generated by the creator.

Creating user account. When a patient wants to use the system, she/he first registers online and runs the Setup algorithm of RBE 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.

Organizing patient’s records. After initializing 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 to 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/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/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. When a specialist wishes to write notes to the GP, the specialist simply encrypts the notes to the role General Practitioner and the GP will be able to read it.

Sharing PHR data. An external organization 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/he runs CreateRole algorithm to create a role for the organization and sends the role secret key to the organization. All the users in an external organization need to obtain the decryption keys either from the patient directly or through the organization. The organization can also request a shared decryption key, and give it to all the users in the organization. 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 organization 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. The criteria for emergency access and the time duration for which the accesses are valid as well as other related legal requirements are likely to vary from country to country. These requirements need to be taken into account in the design of emergency access service. 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. 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. If a patient does not create keys for the Trust Authority, the emergency access to the patient’s PHR data will be denied.

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 organization, such as a hospital, she/he removes the public parameters of any role that is related to the 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 of PCEHR discussed previously have been addressed in 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 organization is referred by another healthcare organization, the referred organization only needs to obtain the decryption key from the patient when she/he registers in the organization 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 organizations 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.

5.2. 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 Fig. 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/he thinks that 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-based letter or document with her when visiting a doctor. All her healthcare information can be accessed by authorized 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.

6. CONCLUSION

In this paper, we have considered a secure cloud healthcare data storage system using a RBE scheme. We first introduced the EHR services system PCEHR provided by the Australian government and reviewed the RBE schemes that are used in our system. We presented our solution for a secure PHR data storage system by describing 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 a EHR storage system such as the PCEHR. We discussed a few weaknesses of the current PCEHR system, and then described our system operations in detail and showed how the weaknesses in the PCEHR system can be addressed by our proposed secure RBE system. In particular, an important aspect is that our system enables the patients’ record data to be stored in encrypted format while the patients still retaining the control of policies in terms of who can access and view their data. We have specified the policy as to who can access them in terms of roles, which is common in many commercial systems including in the healthcare. The enhancements that we have suggested for the PCEHR though they introduce some new functionalities such as encryption and key management operations associated with encryption, they still adhere to the basic principles and ethos of the PCEHR system, which is patient-centric. We believe that the proposed system has the potential to be used easily in commercial healthcare systems due to its flexibility in deployment. It captures practical access policies based on roles in a flexible manner and provides secure data storage in the cloud enforcing these access policies.

REFERENCES


