A PHR System With Policy-Based Fine-Grained Access Control And Revocation Mechanism

by

© *Mitu Kumar Debnath*

A thesis submitted to the

School of Graduate Studies

in partial fulfilment of the

requirements for the degree of

Master of *Science*

Department of *Computer Science*

Memorial University of Newfoundland

October 2015

St. John's Newfoundland

Abstract

Collaborative sharing of information is becoming much more needed technique to achieve complex goals in today's fast-paced tech-dominant world. Personal Health Record (PHR) system has become a popular research area for sharing patients information very quickly among health professionals. PHR systems store and process sensitive information, which should have proper security mechanisms to protect patients' private data. Thus, access control mechanisms of the PHR should be well-defined. Secondly, PHRs should be stored in encrypted form. Cryptographic schemes offering a more suitable solution for enforcing access policies based on user attributes are needed for this purpose. Attribute-based encryption can resolve these problems, we propose a patient-centric framework that protects PHRs against untrusted service providers and malicious users. In this framework, we have used Ciphertext Policy Attribute Based Encryption scheme as an efficient cryptographic technique, enhancing security and privacy of the system, as well as enabling access revocation. Patients can encrypt their PHRs and store them on untrusted storage servers. They also maintain full control over access to their PHR data by assigning attribute-based access control to selected data users, and revoking unauthorized users instantly. In order to evaluate our system, we implemented CP-ABE library and web services as part of our framework. We also developed an android application based on the framework that allows users to register into the system, encrypt their PHR data and upload to the server, and at the same time authorized users can download PHR data and decrypt it. Finally, we present experimental results and performance analysis. It shows that the deployment of the proposed system would be practical and can be applied into practice.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisor Dr. Krishnamurthy Vidyasankar for his guidance, understanding, patience, and most importantly, his continuous support and valuable advice during my graduate studies. I have been fortunate to have him as my advisor who gave me the freedom to explore on my own, and at the same time the guidance to keep myself on the right track. Dr. Vidyasankar taught me how to think independently and I think very few students get the opportunity to develop their individuality and self-sufficiency by being allowed to work with such independence. His patience and support helped me overcome many crisis situations and finish this dissertation.

Second, I would like to gratefully thank my co-advisor, Dr. Saeed Samet, who has been always there to listen to my problems and give advice. He helped me to sort out all kinds of technical details of our work and guided me throughout the research work. His insightful comments and constructive criticisms at different level of the research helped me to keep focus on my ideas. I am indebted to him for his continuous encouragement and guidance.

Finally, It would not be possible for me to come this far without the unconditional love and support from my parents, Monuranjan Debnath and Depali Debnath. They always encouraged me all the way of my life and never stopped believing in me. I thank them from the core of my heart for all their sacrifices to achieve my goals. Last and most importantly, I owe my gratitude to my younger brother, Titu Kumar Debnath, who took care of all the responsibilities on behalf of me and allowed me to concentrate on my studies. His mental support gave me the strength to go forward and stay focused.

I am also grateful to all those people who have helped and encouraged me all the way to complete this thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Cloud computing has transformed our traditional information technology (IT) industry in terms of hardware and software systems. In recent years, *cloud computing* has started emerging to fulfill a long-held dream of computing as a utility. It has introduced every component of IT system as a service such as Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS), etc. It has minimized the cost and improved the efficiency of resource management as well as satisfying business needs on demand. One of the biggest challenges for utilizing cloud computing is ensuring data security and privacy.

The internet has increasingly become popular and widely available in last few years and the next wave of technology shift has come in the form of Internet of Things (IoT) [21] and virtual infrastructure is provided by cloud computing. According to Cisco [24], We are entering into the era of Internet of Everything (IoE) which goes beyond Internet of Things (IoT). All these technological paradigm shifts are to provide seamless experience in collaborative sharing of information. Again, security and privacy issues remain as a challenge.

Currently health sector is gaining huge momentum in adoption of technology in order to accomplish complicated tasks such as identifying diseases and finding cure for some rare diseases. Moreover, it helps doctors and caregivers to provide better treatment and improved patient care. Healthcare providers can easily reach patients and share valuable information with them. Patients can also reach doctors when they need. Technology has enabled doctors to use email, voice and video conference facilities to consult with other doctors from all over the world which is known as telemedicine [9].

Every patient has a health record where information related to the care of a patient is stored. It is known as Personal Health Record (PHR). Personal Health Records (PHRs) are moving from paper-based PHRs to electronic PHRs [20]. It brings numerous benefits in terms of reduced healthcare cost, ease of use, rapid detection and cure of complex diseases, collaborative sharing and exchange of health information, better quality of care, etc. Patient-centric electronic PHRs are being adopted by many healthcare systems. These electronic PHRs are owned and managed by patients themselves and linked with caregivers and doctors in various ways. Data security and privacy issues are very critical, because healthcare data are highly sensitive information. In this thesis, we focus on how patients can share their personal health data securely among doctors and caregivers.

1.2 Motivation

Patients want to get proper care and best treatment available for their health conditions and patient-centric PHR system can help to achieve this goal. PHR system helps providing necessary tools to access, manage, process and share health information in such a way that doctors and caregivers can easily provide treatment and health services to the patients and it specifically helps patients who need long-term treatments, for example patients with chronic diseases. To realize these benefits we have to make sure that data security and privacy issues are resolved in terms of data confidentiality and integrity.

Secure management and sharing a PHR data require proper access control mechanism in place. PHR data must be encrypted before storing with a third party cloud service provider. It should be available only to the users who are given the authorization to access it. Furthermore, patients (owners of the PHR) will hold the right to grant access as well as right to revoke access privileges any time when they feel it is necessary [32]. Therefore, a fine-grained access control along with an efficient revocation mechanism is necessary to protect confidentiality and integrity of patient's health data in an untrusted cloud environment. Key management is a big concern while employing fine-grained access control with revocation mechanism.

In this thesis, we present a practical framework for a PHR system to realize the fine-grained access control with efficient revocation mechanism. To accomplish this goal we use Attribute-Based Encryption (ABE) [40] scheme as our encryption primitive. ABE is a generalization of identity based encryption system that uses attributes to encrypt data for a specific group of users. We implemented the Java library of mCP-ABE [23] scheme, a derivation of ABE, to support our framework design.

1.3 Research outline

We first give a comprehensive overview of PHR system. Next, we examine security and privacy related issues of PHR system. The rest of this thesis is organized as follows:

In chapter 2, we provide background knowledge about PHR systems and discuss in detail one-to-many cryptography schemes such as ABE. We shall see that ABE is quite practical and compares well to other cryptography schemes when addressing PHR related issue. We also discuss related work and compare our work with them in this chapter.

In chapter 3, we provide preliminaries and definitions related to our work.

In chapter 4 and 5, we present our proposed PHR system model and its design framework, and implementation of the PHR framework along with architecture and database design, respectively.

In chapter 6, we present experimental results in order to analyze and validate the practicality of our system. We evaluate the performance of various operations of the CP-ABE library in our PHR system.

Finally, in chapter 7, we summarize our work and discuss the open problems and future work.

Chapter 2

Personal Health Record Systems And Related Work

In this chapter, we define Personal Health Record (PHR), and its properties. We also discuss security and privacy issues related to PHR data sharing among different health professionals. At the end, we present some related work in the area of PHR data sharing and discuss their limitations.

2.1 What is PHR?

According to Markle Foundation [13],

> *"Personal health record (PHR) is a widely used but loosely defined term for a variety of emerging technologies that enable people to manage their health information and health care transactions electronically."*

In other words, PHR allows its owner to store, manage, and share personal health information.

In a PHR system, individuals have full control of their own PHR data. This is the major distinguishing factor that separates PHRs from EHRs (Electronic Health Records). An EHR is the composition of many individual records which are stored in an electronic records system and controlled by the specific health care providers or caregivers. Sometimes, a PHR can be integrated with EHR system and this type of PHR is called patient portal. Here, Our focus is only on PHR systems.

2.2 Attributes of a PHR

According to Markle Connecting for Health [13], an ideal PHR should have the following seven attributes:

- Each person controls their own PHR.
- PHRs contain health information from one's entire lifetime.
- PHRs contain information from all health care providers.
- PHRs are accessible from any place at any time.
- PHRs are private and secure.
- PHRs are transparent. Individuals can see who has entered each piece of data, where it has been transferred from, and who has viewed it.
- PHRs permit easy exchange of information across health care systems.

2.3 Benefits of a PHR system

PHR systems provide several benefits to their users. An individual can store all kinds of health records in one place instead of paper-based files in various doctors' offices. Moreover, patients can share their PHRs with physicians and other health care professionals, which will help them by making better treatment decisions. It also helps analyze an individual's health profile to identify health threats based on their medical history.

2.4 Data security and privacy issues of a PHR system

Despite the benefits and needs of PHR systems, adoption has been low. There might be several reasons for low adoption, but privacy and security issues are one of the biggest concerns [26]. Security is a critical component of a PHR system. People are reluctant to store their sensitive health information into the cloud services owned by, untrusted, third parties, because according to HIPAA, these third parties are not covered entities [36]. Besides, users lose full control of their information once it gets uploaded into the cloud and eventually it violates the first attribute of an aforementioned ideal PHR system.

Moreover, IoT penetrates healthcare sector and it offers greater promises in this field, providing improved access to care, increasing the quality of care and most importantly reduced cost of care [35]. Along with these opportunities a group of brand new security breaches are showing up. According to iHealthBeat.org [37], hackers are directly targeting healthcare organizations. FBI reportedly said that the agency

> *"has observed malicious actors targeting health care-related systems, perhaps for the purpose of obtaining protected health care information and/or Personally Identifiable Information (PII)."*

In a separate incident of massive data breach, an employee from the US Department of Veterans Affairs had stolen personal electronic information of 26.5 million military veterans including their social security numbers, birth dates, and personal health information without authorization [12].

Therefore, a natural way to keep sensitive health data confidential against any untrusted cloud service providers or malicious attackers, is to store only the encrypted data in the cloud. The idea is that PHR system will relinquish full control of data to its owner to encrypt, decrypt and decide with whom to share these sensitive information.

2.5 Related work

Several schemes have been proposed related to data security and privacy of patient health records. Among them Patient Controlled Encryption (PCE) scheme has been proposed by Chase *et al.* in [4]. This scheme allows patients both to share partial access rights with others, and to perform searches over their encrypted records. The scheme uses public key and symmetric key as cryptographic techniques along with their inherent advantages and disadvantages. For example, if the user's secret key is compromised then instant revocation is not available.

Another patient-centric access control (PEACE) scheme has been proposed by Barua *et al.* in [3]. This scheme assures the privacy of patient personal health information (PHI) by defining different access privileges to data requesters and assigning different attribute sets to them. This scheme provides PHI integrity and confidentiality by adopting digital signature and pseudo-identity techniques. It encompasses identity based cryptography to aggregate remote patient PHI securely. However, existing schemes have several limitations, such as lack of fine grained access control and efficient revocation mechanisms.

There are other PHR systems, which use role-based access control (RBAC) scheme to manage access to the user PHR data. Users need to trust these systems for the security of their PHR data. If servers are compromised then users' data will be at risk. Indivo [1] PHR system is a platform, which lets its user to control and manage their data but the access control system is enforced by the Indivo servers. Moreover, commercial cloud-based health record systems such as Practice Fusion [14] and CareCloud [7] are also based on access control decisions defined by their respective cloud storage providers. Microsoft's HealthVault [22] and discontinued Google Health are both cloud-based PHR services providing end-to-end encryption during data transmission from patient to the storage server and vice versa. They also rely on access control mechanism instead of full encryption of PHR data.

iHealthEMR [2] uses Attribute-Based Encryption scheme for a self-protecting Electronic Medical Records (EMRs). Medical records are XML-based files. So, each node of the XML-based file is evaluated by a policy engine to determine if encryption is necessary. patient can encrypt XML-based EMR file with an automatic generated access policy before uploading to cloud storage server. Attributes within the private key defines user's access to the file. However, key management and effective revocation remain as a critical issue.

Attribute Based Encryption (ABE) is first introduced by Sahai *et al.* in [40] as fuzzy Identity-Based Encryption (IBE), a new type of IBE. In this encryption scheme, an identity is a set of descriptive attributes. A private key with a set of attributes, ω can decrypt a ciphertext encrypted with an identity, ω_0, if and only if $|\omega \cap \omega_0| \geq d$, where d is the error-tolerance in terms of minimal set overlap. After that two variants of ABE scheme are introduced, Key-Policy Attribute-Based Encryption (KP-ABE), and Ciphertext-Policy Attribute-Based Encryption (CP-ABE) in [19]. Waters *et al.* in [5] proposed the construction of CP-ABE scheme under the generic group model and it was the first construction under this model.

Key management is a real problem in cryptographic solutions and attribute revocation is one of the biggest issues in designing an effective CP-ABE scheme, addressed in [38]. This paper proposed to use an expiration time for attributes so that they are valid until time gets expired. The problem of this type of approach is that attributes are not revoked until time expires.

Chapter 3

Preliminaries And Definitions

In this chapter, we will describe the terminologies, techniques, and concepts used in our system to make sure that reader will have basic understanding of our system design.

3.1 Pairing-Based Cryptography

At the very beginning pairings were used as cryptanalytic tools to study and analyze cryptographic systems in order to understand how they work and find flaws that will allow them to be broken with or without the key. Pairings were first used in cryptography to reduce the complexity of the discrete logarithm problem on some elliptic curves. After that cryptographers have seen a revolutionized use of pairings through the works of Antoine Joux [25]. In 2000, Joux has shown that Weil and Tate pairings can be used to build a tripartite generalization of the Diffie-Hellman protocol. The world of cryptography spurred with huge potential of pairing-based cryptosystems. After that pairings have started their journey as a building block for various cryptographic schemes such as identity-based schemes, key agreement protocols, attribute-based schemes, and various signature schemes.

Table 3.1: Notations for pairings

Notations	**Description**
$\mathbb{G}$	A finite cyclic group on elliptic curve.
g	g is a generator of cyclic group, $\mathbb{G}$.
$\langle g \rangle$	g is a generator of cyclic group, $\mathbb{G}$ with prime order r. Here r denotes the size of the group.
$\mathbb{F}_q$	$\mathbb{F}$ is the finite field of order q. q denotes the size of the field.
$\mathbb{G}_1$, $\mathbb{G}_2$	These are two cyclic groups of points on the curve.
$\mathbb{G}_T$	It is a subgroup of the multiplicative group of a related finite field, $\mathbb{F}^*_{q^k}$. Here k is a parameter called the *embedding degree* in pairing-based cryptography.

3.1.1 Elliptic curves

Elliptic curves have brought great advantages in the field of cryptography in terms of smaller key size, higher computational speed, lower bandwidth consumption, etc. Elliptic Curve Cryptography (ECC) [6, 27, 34] is well-suited for resource constrained systems such as smart cards, smartphones, tablets, notebooks, and so on. ECC is gaining popularity as a public key cryptography especially for resource constrained wireless systems. But this is another topic of discussion and one can get a glimpse of this idea from [42].

In 1985, Koblitz [27] and Miller [34] separately suggested that discrete logarithm-based cryptosystems can provide better security by using the group of points on an elliptic curve over a finite field rather than the conventional multiplicative group of a finite field. The reason for better security over systems based on multiplicative group of a finite field and systems based on the intractability of integer factorization is the absense of a subexponential-time algorithm that could find discrete logarithms in these groups [28].

3.1.2 Type A curves

There are six types of supersingular curves, Type A, Type A1, Type D, Type E, Type F, and Type G [30]. In our case, we select Type A curves to build pairings.

Definition 3.1.1. *Type A curves* are supersingular curves constructed on the curve $y^2 = x^3 + ax$ over the finite field $\mathbb{F}_q$, for any $a \in \mathbb{F}_q$ and for some prime $q = 3 \ mod \ 4$, with embedding degree of 2.

3.1.3 Cyclic groups

Cyclic groups are of two kinds: finite and infinite. We are more interested in finite groups because pairings are based on finite cyclic groups. More about groups and fields can be found in [18].

Definition 3.1.2. A finite group $\mathbb{G}$ is *cyclic* if there is $g \in \mathbb{G}$ so that $\langle g \rangle = \mathbb{G}$. And such a g is a generator of $\mathbb{G}$, and $\mathbb{G}$ is said to be generated by g.

Finite cyclic groups are suitable for cryptosystems because of their easy group operations and the intractability of discrete log problem. Cyclic groups have several properties:

- Subgroups of cyclic groups are cyclic.
- A cyclic group has a single generator.
- All cyclic groups are abelian.
- Every finite group of prime order is cyclic.

3.1.4 Bilinear maps

Generally, two types of pairings are used in pairing-based cryptosystems. One is called *asymmetric pairing*:

$$\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T \tag{3.1}$$

In this eq. (3.1) $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic groups of prime order n. Another type is the special case which is called *symmetric pairing* where $\mathbb{G}_1 = \mathbb{G}_2$.

$$\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T \tag{3.2}$$

However, $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive groups whereas $\mathbb{G}_T$ is a multiplicative group. Let g be the generator of group $\mathbb{G}_1$. Now we will give a formal definition of (symmetric) bilinear pairing or mapping in the context of pairing-based cryptography.

Definition 3.1.3. A *bilinear mapping* or *pairing* on cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_T$ is a map, $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ which satisfies the following conditions:

1. *Bilinearity*: $\hat{e}(g^a, g^b) = \hat{e}(g, g)^{ab}$ for all $a, b \in \mathbb{Z}$.

2. *Nondegeneracy*: $\hat{e}(g, g) \neq 1$.

3.1.5 What is pairing?

Pairing in cryptography is the mapping of one group to another with some special properties. In other words, a pairing on two cyclic groups, $\mathbb{G}_1$ and $\mathbb{G}_2$, based on an elliptic curve is simply a special function $\hat{e}$ which takes pairs of elements from $\mathbb{G}_1$ as inputs and produces an element of group $\mathbb{G}_2$. Pairing among $\mathbb{G}_1$ and $\mathbb{G}_2$ is denoted as:

$$\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2 \tag{3.3}$$

The pairing has some special properties and *bi-linearity* is the most important of them. However, all of the pairings are based on either Weil or Tate pairing on elliptic curves over finite fields [17]. In our system design Pairing-Based Cryptography (PBC) [31] library uses Tate pairing. Tate pairing is well defined and its actual computation is simple and easier [30, 33].

Tate pairing [16, 30] Let E be an elliptic curve containing n points over a field $\mathbb{F}_q$. Let $\mathbb{G}$ be a cyclic subgroup of $E(\mathbb{F}_q)$ of order r with r, q co-prime. Let k be the smallest positive integer such that the field K (For brevity, $K = \mathbb{F}_{q^k}$), the smallest extension of $\mathbb{F}_q$, contains the r^{th} roots of unity (in other words, $r | q^k - 1$). $E[r]$ is the subgroup of points of order r and $E(K)/rE(K)$ is the quotient group. Then the Tate pairing is a mapping:

$$e : E[r] \cap E(K) \times E(K)/rE(K) \rightarrow K^*/K^{*r} \tag{3.4}$$

The quotient group on the right hand side of Equation (3.4) can be thought of as the set of equivalence classes of $\mathbb{F}_{q^k}^*$ under the equivalence relation $a \equiv b$ if and only if there exists $c \in \mathbb{F}_{q^k}^*$ such that $a = bc^r$. This relation is called 'equivalence modulo r^{th} powers'.

Tate pairing is non-degenerate and bilinear. In cryptography, pairings are built using cyclic groups on elliptic curves and Tate pairing on supersingular elliptic curve is desired in pairing-based cryptosystems [15].

3.2 Cryptographic techniques

There are well established and efficient traditional cryptographic systems available for data encryption and decryption but none of them quite fit in the context of PHR systems. In a PHR system, a user can encrypt data to share selectively with a group of users and at the same time authorization to access data for a particular user can be revoked by the owner.

Symmetric-Key Encryption (SKE) such as Advanced Encryption Standard (AES) [41] and Public-Key Encryption (PKE) such as RSA [39] are secure encryption schemes but they do not provide fine-grained access control mechanism for secure PHR systems. The reason is that they use one-to-one encryption methods which cause low scalability in the presence of large user base of PHR systems. Key management is also a big issue. On the other hand, attribute-based encryption scheme follows one-to-many encryption method which is suitable for PHR systems. It provides fine-grained access control mechanism in terms of attributes and access policy [5]. One important feature of ABE is that it is *collusion-resistant* [40].

3.2.1 Attribute-Based Encryption

3.2.1.1 Definition

Attribute-Based Encryption (ABE) is a public-key encryption scheme where each encrypted item is associated with a policy. In this system, a user's keys and ciphertexts are labelled with sets of descriptive attributes and a particular key can decrypt a particular ciphertext only if there is a match between the attributes of the ciphertext and the user's key.

3.2.1.2 Overview

Attribute-based encryption is the generalization of Identity-Based Encryption (IBE) system [40] and it has potential for providing data security in the context of distributed environment.

To provide data security and privacy traditional Public Key Infrastructure (PKI) is very useful but it has several drawbacks which are not suitable for some particular environments. For example, if a user wants to send a message to a group of recipients then she needs to encrypt the same message again and again with different public keys in order to send it to the respective different recipients and this becomes impractical when the number of recipients increases. Moreover, every public key needs to be verified against a Certificate Authority (CA) to check the authenticity of its owner.

Identity-based encryption system resolves the problem of verifying public keys. In IBE a user's identity is a string which could be her email address, for example, Alice's email address is "alice@mun.ca" and this identity becomes her public key. Whoever wants to send her message can use her email address as a public key and no need to access CA for public key verification. But scalability issues remain the same because IBE uses one-to-one communication method similar to traditional PKI system. There are some application areas where data will be encrypted with a specific policy and will be shared without prior knowledge of recipients' identities. For this particular type of setting, attribute-based encryption scheme comes into the picture.

Pirretti *et al.* in [38] clearly describe the structure of attribute-based encryption scheme. In attribute-based encryption a user's identity is composed of a set, S, of strings which serve as descriptive attributes of the user. For example, a user's identity could consist of attributes describing their organisation, department, expertise, and job function. A user in the system can then specify another set of attributes S' such that a recipient can only decrypt a message if her identity S has at least k attributes in common with the set S', where k is a threshold parameter set by the system. So the user's description, a set of attributes, becomes the public key. However, the expressiveness of attribute-based encryption system is much more powerful because several different recipients could possess k attributes in common who are able to decrypt a message encrypted for a set S'.

Attribute-based encryption has two flavours: Cipher Policy Attribute-Based Encryption (CP-ABE) and Key Policy Attribute-Based Encryption (KP-ABE).

KP-ABE In this encryption scheme, private keys are associated with access policy and ciphertexts are labelled with sets of attributes. In this scheme private keys determine which ciphertexts are to be decrypted.

CP-ABE This scheme is quite opposite of what KP-ABE scheme does. In this scheme ciphertexts are associated with access policy and private keys are associated with sets of attributes.

In our setting, we will use CP-ABE scheme as encryption scheme because data owner wants to have control over her sensitive data and wants to decide who can access that PHR data. In contrast, encryptor in KP-ABE scheme has no control over who can access her encrypted data. She has to trust the authority who generates keys, this is not suitable for our patient-centric PHR system.

Table 3.2: Notations for access tree

Notations	**Description**
$\mathcal{T}$	Representing access policy structure as access tree with root r
$\mathcal{T}_x$	Subtree of $\mathcal{T}$ rooted at node x
num_x	The number of children of a node x
k_x	Defines threshold value at node x
$parent(x)$	Parent of the node x in the access tree $\mathcal{T}$
$attr(x)$	Denotes attribute associated with the leaf node x in the access tree
$index(x)$	Index of the x's child nodes
S	Defines a set of attributes

3.2.2 Access policy structure

Access policy structure [5] in ABE is a tree based on a boolean function consisting of (OR, AND) gates between attributes. A user can decrypt a ciphertext successfully provided that the attribute set of his secret key satisfies the access policy associated with the ciphertext.

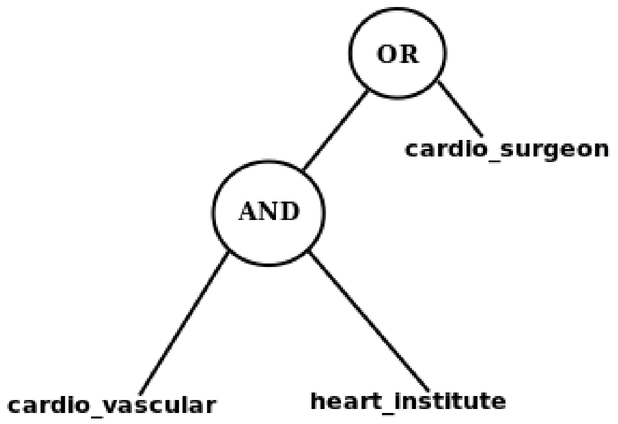


Figure 3.1: A structure of access policy

Figure 3.1 shows an access policy with three attributes "cardio_surgeon", "cardio_vascular", and "heart_institute". If a user's secret key holds either "cardio_surgeon" attribute or a combination of "cardio_vascular" and "heart_institute" attributes then this access policy will be satisfied and decryption process can take place.

Access tree [5] Let $\mathcal{T}$ be a tree representing an access structure. Each non-leaf node of the tree represents a threshold gate, described by its children and a threshold value. If num_x is the number of children of a node x and k_x is its threshold value, then $0 < k_x \leq num_x$. When $k_x = 1$, the threshold gate is an OR gate and when $k_x =$

num_x , it is an AND gate. Each leaf node x of the tree is described by an attribute and a threshold value $k_x = 1$.

Table 3.3: Notations used in our PHR framework

Notations	**Description**
$\mathcal{U}$	The universal or system attribute set
PK	Public key or public key parameters
MK	Master key, which is kept secret by Trusted Authority (TA)
I_u	It denotes each registered user of the system with a unique identifier.
SK	Secret key generated based on a set of user attributes, S.
$SK_{I_u,1}$	First part of the secret key generated for a specific user I_u.
$SK_{I_u,2}$	Second part of the secret key generated for a specific user I_u.
CT	It denotes any ciphertext generated by encryption algorithm of our system.
$\hat{CT}$	It denotes an intermediary ciphertext or a decryption token generated by revocation server.
M	It denotes the plaintext before the encryption and after the decryption operation.

Satisfying an access tree [5] Let $\mathcal{T}$ be an access tree with root r. $\mathcal{T}_x$ is the subtree of $\mathcal{T}$ rooted at the node x. Hence $\mathcal{T}$ is the same as $\mathcal{T}_r$. If a set of attributes S satisfies the access tree $\mathcal{T}_x$, we denote it as $\mathcal{T}_x(S) = 1$. We compute $\mathcal{T}_x(S)$ recursively as follows. If x is a non-leaf node, evaluate $\mathcal{T}_{x'}(S)$ for all children x' of node x. $\mathcal{T}_x(S)$ returns 1 if and only if at least k_x children return 1. If x is a leaf node, then $\mathcal{T}_x(S)$ returns 1 if and only if $attr(x) \in S$.

3.2.3 Ciphertext Policy Attribute-Based Encryption (CP-ABE)

In our attribute-based encryption scheme we choose Ciphertext Policy Attribute-Based Encryption (CP-ABE) [5]. In this scheme, user's secret keys are associated with a sets of attributes, whereas ciphertexts are associated with the policies. Generally, CP-ABE consists of four procedures: Setup, Encrypt, KeyGen, and Decrypt.

- **Setup:** The setup algorithm doesn't take any input other than the implicit security parameter.[1] It outputs the public key PK and a master key MK.
- **Encrypt**$(PK, M, \mathcal{T})$**:** The encryption algorithm takes as input the public parameters PK, a message M, and an access structure $\mathcal{T}$ over the universe of attributes. The algorithm will encrypt M and produce a ciphertext CT such that only a user that possesses a set of attributes, S that satisfies the access structure will be able to decrypt the message.
- **Key Generation**(MK, S)**:** The key generation algorithm takes as input the master key MK and a set of attributes S that defines the key and outputs a private key SK.
- **Decrypt**(PK, CT, SK)**:** The decryption algorithm takes as input the public parameters PK, a ciphertext CT, which contains an access policy $\mathcal{T}$, and a private key SK, which is a private key for a set S of attributes. If the set S of

[1]Security parameter is a Type A curve parameter, which is used to create pairings in CP-ABE scheme to generate master key and public key.

attributes satisfies the access structure $\mathcal{T}$ then the algorithm will decrypt the ciphertext and return a message M.

Discussion We discussed Pairing-Based Cryptography and its different elements. We also described different types of pairings and their properties. We use Attribute-Based Encryption (ABE) scheme in our PHR framework and it is based on Pairing-Based Cryptography. We also discussed about other traditional encryption schemes and argued that Attribute-Based Encryption scheme is suitable in the context of PHR framework. At the end, we describe CP-ABE scheme, a variant of ABE scheme, as the encryption scheme for our framework.

Chapter 4

Proposed System

4.1 Models, assumptions, and requirements

We will first describe general system structure and security model of our PHR framework and then we will discuss security requirements of the system followed by design goals. Moreover, all the terminologies and notations used in our system design are described in table 4.1 and table 3.3, respectively.

4.1.1 System models

Our PHR framework consists of six entities, as follows:

- **Authorization Server (AS):** An entity which is responsible for handling the registration, login, and logout process to the system. It authorizes the access of users to the PHR system.

- **Trusted Authority (TA):** An entity which is trusted by all other participating entities in this system. It is trusted in the sense that it securely generates and

Table 4.1: Terminologies

Terms	**Description**
PHR Data	We also mention it as data file, PHR information, PHR file, or simply data. The internal structure of PHR data is somewhat abstract in our our context.
Storage Provider	We also mention it in this thesis as cloud provider, service provider, cloud service provider, or simply as cloud, etc. as untrusted storage providers.
Data Owner	It refers to patient, caregiver, doctor, etc. who own data.
User	User could be anyone who is authorized in the system and can access and consume data.

stores master key and users' secret keys and securely transmits those secret keys to the users upon valid requests.

- **Revocation Server (RS):** A semi-trusted entity which will issue decryption tokens to the users. It also maintains an Attribute Revocation List (ARL) based on revoked users and attributes. It is semi-trusted in the sense that it correctly follows the protocol, but might also try to learn the intermediate information.

- **Data Owner:** The entity who owns data and encrypts those data (*e.g.,* patients).

- **User:** The entity who would like to access encrypted data with proper authorization. Users are based on hierarchical structure. Basically, we divide our users into two groups; patients and health professionals. Patients are not in the hierarchy but health professionals come from different levels of hierarchy such as a surgeon can come from the surgery department of health science centre in St. John's area. We will describe the details of hierarchical structure of attributes in next chapter.

- **Storage Provider:** The entity who will provide storage service to store encrypted data.

The data owner encrypts their PHR data first and then uploads them to the cloud servers. The appropriate user can download desired data file from cloud and decrypt them with proper key before accessing the content of that file. The data owner also creates access policy based on the universal set of attributes. We assume that storage providers, *i.e.* cloud servers, are always online and have adequate storage capacity along with high computing power. We also assume that revocation server will be always online and will not provide any decryption token to the revoked attributes based on revocation list. TA will also be online to revoke the system and user attributes, to generate and distribute secret keys to users. TA is also responsible to maintain the secrecy of the master key.

4.1.2 Security models

We consider cloud servers (most of them) are not trustworthy in the sense that they will try to access the content of PHR data files. In some cases they will also try to collude with malicious users to get access to the content of data file. Moreover, some users might try to get unauthorized access to those encrypted data files. These malicious or unauthorized users can work independently or together to gain access to these sensitive contents. Each user of the system will have a secret key based on her designated attribute set and public parameters will be available to everybody.

End-to-end encryption Communication between users and cloud providers are assumed to be secured under existing security protocols such as HTTPS/TLS [11]. The idea behind this end-to-end encryption is to maintain data integrity along with data confidentiality during communication process.

4.1.3 Security requirements

The security requirements of our PHR system are:

- **Data Confidentiality:** Preserving the confidentiality of user data is a key requirement for a PHR system. Data should be accessible to only those who are explicitly authorized by the data owner.

- **Data Integrity:** Integrity of the data must be ensured so that PHR users can be certain that content posted to a third party storage server is authentic.

- **Instant Revocation:** If a user's attribute is expired or revoked, then immediately the user should not be able to decrypt and access any PHR data using that attribute, known as *attribute revocation*.

- **Collusion-resistance:** Multiple users (cloud providers also) will try to collaborate with each other but they should not be able to decrypt the ciphertext by combining their secret keys. This is one of the strongest security requirements for our system.

Data availability is another requirement for privacy aware PHR system which means data should remain available despite any disaster unless explicitly deleted by its owner, and despite potential malicious attempts to destroy data. However, this security requirement relies more on the nature of network infrastructure which is out of the scope of our work for now.

4.1.4 Design goals

The main goal of our framework is to provide secure PHR systems with fine grained access control and efficient key management at the same time. We also want to enable data owner and TA to revoke users/attributes instantly with the help of revocation server. In this section, we describe our patient-centric secure data sharing framework for cloud-based PHR systems.

4.1.5 Problem definition

We consider a PHR system with multiple data owners and users. The data owners refer to the patients who have full control over their own PHR data, *i.e.* they can create, manage, update, and delete it anytime. Any third party cloud service provider can store the owners' encrypted PHRs. The users may come from various aspects; for example, a caregiver, a doctor or a nurse. Users may access the PHR documents through the server in order to read someone's PHR, and a user can simultaneously have access to multiple owners' data provided that the user has sufficient secret keys to access those data.

We will describe briefly the CP-ABE scheme that we will use in our system design. We will also mention about the security assumptions for that scheme.

4.1.6 Mediated Ciphertext-Policy Attribute-Based Encryption (mCP-ABE)

The mCP-ABE [23] scheme consists of five algorithms: Setup, Keygen, Encrypt, m-Decrypt, and Decrypt. However, Setup and Encrypt algorithms are the same as original CP-ABE scheme. So here we will only describe three other algorithms.

- **Keygen**(MK, S, I_u)**:** TA runs this algorithm by taking master key MK, user attribute set S, and user identifier I_u as input and generates two secret key shares associated with user attribute set S, and user identifier I_u as output. The first share of the secret key $SK_{I_u,1}$ is delivered to the revocation server, and the second share of the secret key $SK_{I_u,2}$ is delivered to the user. The secret key shares are delivered through a secure channel to the revocation server and to the user.

- **m-Decrypt**$(CT, SK_{I_u,1}, I_u)$**:** Revocation server runs this algorithm by taking ciphertext CT, user identifier I_u, and secret key part $SK_{I_u,1}$ as input and generates intermediate message $\hat{CT}$ as output. If non-revoked attributes from the set S do not satisfy the access tree $\mathcal{T}$ then it generates error symbol $\perp$.

- **Decrypt**$(\hat{CT}, SK_{I_u,2}, CT)$**:** User's client machine runs this algorithm by taking intermediate ciphertext $\hat{CT}$ which is sent from revocation server, secret key part $SK_{I_u,2}$, and original ciphertext CT as input and generates original plaintext message M as output. Again if non-revoked attributes from the set S do not satisfy the access tree $\mathcal{T}$ then it generates error symbol $\perp$. We explain in Section 4.1.8.3 the reason for checking again if access policy is satisfied in client side decryption.

4.1.6.1 Attribute and its hierarchy

Attributes define and classify data, enable efficient identification and classification of similar objects. For example, a group of health professionals with some common interest or responsibilities in a health organization can be segregated based on a set of attributes, such as organization, department, profession, etc. These attributes are then used to build access permission to the specified individuals.

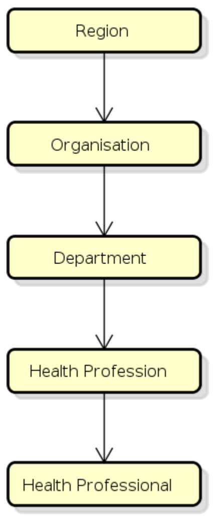


Figure 4.1: A hierarchy of attributes in PHR system

Attribute-based systems have huge potential for providing data security in health sector. For example, a patient Alice with heart problems wants to share her PHR data with appropriate personnel to get proper advise can encrypt her PHR data with the attributes “cardio_surgeon” and “heart_institute”. Only health professionals satisfying these attributes would be able to decrypt this information and can provide health support.

In our system, we created hierarchy-based attributes. Figure 4.1 illustrates the hierarchy of our system attributes. The attribute hierarchy is extendable. In this hierarchy we have five levels of attributes to define, classify, and identify the data. Patients can encrypt their data combining various levels of attributes which enhances flexibility and expressiveness of access policy for any PHR data. For example, a patient can define access policy for her PHR data such that any cardiac surgeon of the surgery department from the heart institute in St. John’s can access PHR data. In this case, four level of attributes are being used to identify authorized health professionals namely region, organisation, department, and health profession attributes.

4.1.6.2 mCP-ABE implementation

We have implemented the Java library for mCP-ABE scheme as a part of our framework design. We used jPBC [10] library for the underlying pairings and other pairing-based operations. This is the Java porting of original Pairing-Based Cryptography (PBC)[1] [31] library which is designed and written in C by Ben Lynn. The original CP-ABE scheme which is designed by Waters *et al.* in [5] is written in C language also. We have implemented our own Java implementation of mCP-ABE library. To the best of our knowledge, this is the first mCP-ABE library implementation based on the work

[1]Pairing-based cryptography is the use of a pairing between elements of two cyclic groups to a third group. The detail description about PBC can be found in Chapter 3.

of Ibraimi *et al.* in [23]. We will describe the details of this library implementation in chapter 5.

4.1.7 Complexity assumptions for security

We already know that all the Identity-Based Encryption (IBE) and Attribute-Based Encryption (ABE) schemes are based on PBC and the security of PBC scheme along with other cryptosystems that use pairings hugely depends on the intractability of (Bilinear) Diffie-Hellman problem, discrete logarithm problem, or other related problems. This mCP-ABE scheme is also based on the fact that the discrete logarithm and the Diffie-Hellman problem are hard to solve as long as the order of the group is a large prime number.

Definition 4.1.1. Let $\hat{e}$ be a bilinear pairing on $(\mathbb{G}_1, \mathbb{G}_2)$. The bilinear Diffie-Hellman problem (BDHP) is the following: Given P, aP, bP, and cP compute $\hat{e}(P, P)^{abc}$.

4.1.8 Proposed PHR framework

We are using CP-ABE as the building block of our PHR system to gain fine-grained access control with providing data confidentiality.

The user secret key in our framework gets divided into two parts. One part of the secret key goes to the user and another part goes to the revocation server. To decrypt the PHR data, the user has to get the decryption token from the revocation server, which is depicted in Chapter 5. The revocation server keeps an Attribute Revocation List (ARL) and refuses issuing the decryption token for any revoked attribute. When a user wants to decrypt a ciphertext, if user's attributes are in ARL then the revocation server will not produce any token and decryption will not take place, which implies instant revocation of user attributes.

We consider our framework in three different parts. These are:

- System setup and key distribution
- PHR encryption and access
- Revocation mechanism

We show the overview of the framework design in Figure 4.2 and the details of the system design and implementation are presented in Chapter 5.

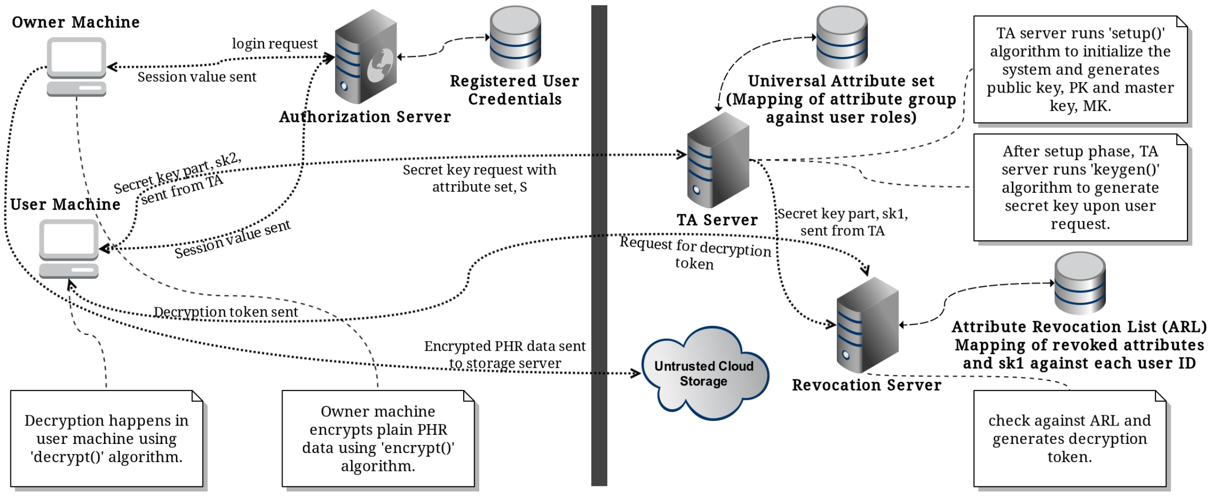

Owner Machine
login request
Session value sent
Authorization Server
Registered User Credentials
Universal Attribute set (Mapping of attribute group against user roles)
TA server runs 'setup()' algorithm to initialize the system and generates public key, PK and master key, MK.
After setup phase, TA server runs 'keygen()' algorithm to generate secret key upon user request.
Secret key part, sk2, sent from TA
User Machine
Secret key request with attribute set, S
TA Server
Secret key part, sk1, sent from TA
Session value sent
Request for decryption token
Decryption token sent
Encrypted PHR data sent to storage server
Untrusted Cloud Storage
Revocation Server
Attribute Revocation List (ARL) Mapping of revoked attributes and sk1 against each user ID
Decryption happens in user machine using 'decrypt()' algorithm.
Owner machine encrypts plain PHR data using 'encrypt()' algorithm.
check against ARL and generates decryption token.

Figure 4.2: An overview of the PHR framework

4.1.8.1 System setup and key distribution

It is crucial to define a proper set of attributes for the PHR system. The system first defines a common universe of attributes $\mathcal{U}$, such as "surgeon," "Health Science

Center," "allergies," "prescriptions," "nurses from surgery," "ICU doctors," etc. The system defines attributes, and users of the system obtain secret keys from TA, which binds the users to their claimed attributes. There would be a process in the TA to map an authorized personnel with a subset of attribute set based on her designated role. For example, a doctor might receive "emergency center, cardiac surgeon, internal medicine" as her attribute set from the TA. So if a nurse requests for a secret key based on this attribute set then she might get refused by the TA, because her designation doesn't allow her to request a secret key based on the attribute set which are only allowed for doctors.

4.1.8.2 PHR encryption and access

A data owner, *i.e.* a patient, who wants to encrypt PHR data using a particular access policy, first checks for all valid attributes from the universal set $\mathcal{U}$ and creates access policy based on those valid attributes. The owner then encrypts her PHR data with this access policy and stores it to the server. Figure 4.2 shows that owner machine encrypts the data and uploads it to the cloud. Only authorized users can decrypt these PHR documents. An authorized user, *i.e.* a health professional, downloads PHR documents from the server to the user machine (Figure 4.2) in order to access the content of the document. The user must request her secret key part, SK_2 before the decryption process takes place. The user can decrypt PHR document if and only if she has suitable secret key part which would satisfy the corresponding access policy of the encrypted file and for which attributes are not in the revocation list of the revocation server. However, the owner can change her access policy for a particular PHR data and re-encrypt that file again with a new policy.

4.1.8.3 Revocation mechanism

The revocation server is assigned to filter out all revoked attributes and users from being served with their decryption requests. To accomplish this task the revocation server maintains an Attribute Revocation List (ARL). When an attribute gets revoked from the universal set $\mathcal{U}$, TA removes it from the universal attribute set and requests revocation server to list up that attribute in the revocation list and not to serve further requests based on this attribute. When an attribute is revoked from a particular user's attribute set, which is mapped against each user ID I_u, TA requests revocation server to enlist that attribute against user I_u into the revocation list and not to perform any request for that attribute, which comes from that particular user I_u. In this way, revocation gets enforced instantly. However, as we mentioned before, either data owner or TA holds the authorization to decide which attribute to be revoked. These operations are done by the TA or revocation server. From above description we can identify several possible cases for revocation:

- Revocation of one or more attributes from the universal set $\mathcal{U}$ which will affect the entire system.
- Revocation of one or more attributes from a user attribute set S which affects only that particular user.
- Revocation of a user which is equivalent to revoking all requests from that user.

In our scheme, decryption of the ciphertext is not fully performed on revocation server rather decryption is performed in two levels instead of one level. As it is a patient-oriented system, in almost all steps that are directly related to data access, we want to involve the patient's permission and involvement. Although the revocation server is a semi-trusted party, *i.e.* it correctly follows the protocol steps, it might curiously misuse the intermediate information. If the complete control is given to the this server, the patient's private data might be compromised. As the revocation server keeps the Attribute Revocation List (ARL), data users do not need to contact data owners every time they want to get access to the data they are authorized to.

Usually, data owners do not have high computation power on their devices to do extra tasks related to data access through the network. Therefore, revocation server will partially handle some tasks on behalf of the data owners. Data owners are not online all the time, while revocation server is. This feature will make the system more practical in real-world applications. Having all of the above items, although the system might need to have some redundant and extra tasks and components, it will be robust, secure and applicable.

Discussion To sum up, we point out some key features that our framework achieves to provide data security and privacy in the context of PHR system. These are:

1. It provides data confidentiality by encrypting data before uploading to the cloud. In this way, unauthorized users including storage service providers cannot access the content of PHR files.

2. Fine-grained access control is achieved using our specific CP-ABE scheme and data integrity is preserved by end-to-end encryption system.

3. It is collusion-resistant meaning that malicious users cannot decrypt a particular encrypted file combining their attributes and secret keys as well. To prevent

collusion, the Keygen algorithm of CP-ABE scheme generates a random value u_{id} for each user I_u. This random value is embedded in each component of the user secret key. Users cannot combine components of the secret key to increase decryption power since different users have different random value in their secret keys.

4. The sizes of the ciphertext and secret key do not increase linearly with the total number of attributes in the system. The size of the secret keys depends on the number of the user's attributes and the size of the ciphertext depends roughly on the size of the access policy.

5. Instant revocations of users or attributes are achieved through our framework.

We also point out some limitations of our framework. These are:

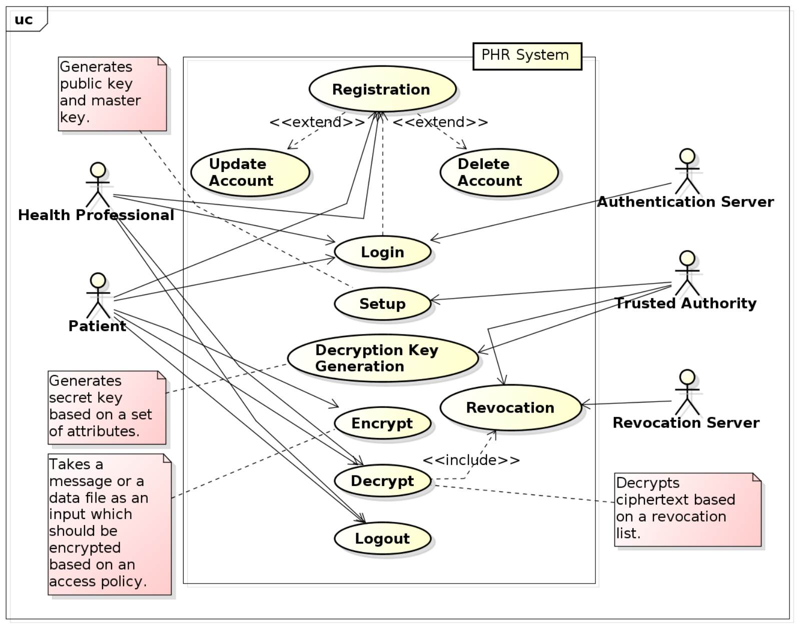
1. System needs to generate different secret keys for each user based on user's attribute set.

2. Data owner may not know the attributes of a specific user. Owner needs to know attributes of the intended users before encrypting data for them.

Chapter 5

Design And Implementation

At first we describe the whole system and its functional overview using UML diagram and then gradually we go in details of the system design. Figure 5.1 describes the overview of PHR system.



uc
PHR System
Generates public key and master key.
Registration
<<extend>>
<<extend>>
Update Account
Delete Account
Health Professional
Authentication Server
Login
Setup
Trusted Authority
Patient
Decryption Key Generation
Generates secret key based on a set of attributes.
Revocation
Revocation Server
Encrypt
Takes a message or a data file as an input which should be encrypted based on an access policy.
<<include>>
Decrypt
Decrypts ciphertext based on a revocation list.
Logout

Figure 5.1: An overview of PHR system design

5.1 Architecture

The proposed framework would have a Cloud Provider (CP) for third party storage, a Private Secured Server (PSS) for TA and/or mediator, and users' own devices. We will use a server instance, e.g. PSS, to run the PHR system. An instance of Windows or LINUX server can be used to run the application. For the purpose of this project we have deployed the system on LINUX instance. We have used MySQL relational database model to store application related data. The client application has been used to encrypt and decrypt PHR data and then stored on a cloud Storage Service.

The PSS is initialized by running the setup function of CP-ABE. The function creates the Master Key (MK) and the Public Key (PK). The MK will be saved privately in the PSS without revealing it to other users. On the other hand, the PK will be available for users to be able to encrypt and decrypt data. Here PSS can work as a TA to create private keys for each registered user with identifiable and valid attributes.

The PHR system has been built on RESTful[1] web services and is residing in instance of a server. Using Java web services, CP-ABE libraries are accessed and encryption and decryption tasks take place. Web services are platform independent and thus different types of client application can be developed based on those web services. All encrypted PHR files will be stored on the cloud server. MySQL database would store all the metadata and references related to PHR files, user credentials, and attributes assigned to users. Each user will have a different set of attributes. Whenever a user demands a file, using the attributes assigned to the user, decryption will take

[1]Representational State Transfer (REST) is an architectural style for web services, where data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs).

place.

5.2 Implementation

Our implementation has four different parts, as follows

- CP-ABE library implementation
- Web services for framework
- Database system of the framework
- Android client application

5.2.1 Java implementation of CP-ABE library

All variants of ABE schemes are based on pairing-based cryptography. Our CP-ABE library implementation is also supported by Java porting of PBC library named jPBC. PBC library is the only complete implementation of bilinear maps in cryptography. Java programming language is widely used language for developing any type of systems ranging from enterprise levels to mobile devices. We have chosen Java to implement our CP-ABE library.

jPBC supports different types of elliptic curves. These are Type A, Type A1, Type D, Type E, Type F, Type G. We used Type A curves in our implementation. Type A pairing is symmetric, with bilinear pairings and our CP-ABE scheme is based on bilinear mapping. That's why, we used Type A pairing as the building block for our library implementation. Table 5.1 shows the pairing time and other pairing-based operations in milliseconds. The test has been done in Google Nexus 4 with Android OS 5.1.1 and Qualcomm Snapdragon S4 Quad-core 1500 MHz Processor. The run time is including the system time.

Table 5.1: Pairing operation in milliseconds

Pairing Type	**Average pairing time**	**Average preprocessed pairing time**
a	853.0	340.0
d159	7265.0	6707.0
d201	7743.0	7763.0
d224	9821.0	9764.0

Creation of pairings Type A pairings are constructed on the curve $y^2{=}x^3{+}x$ over the field $\mathbb{F}_q$ for some prime $q{=}$ 3 mod 4. Both $\mathbb{G}_1$ and $\mathbb{G}_2$ are the group of points $E(\mathbb{F}_q)$, so this pairing is symmetric.

```
import it.unisa.dia.gas.jpbc.*;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

Pairing pairing = PairingFactory.getPairing("a.properties");
```

Figure 5.2: Initialization of pairing

The Pairing interface from jPBC library provides access to the algebraic structures underlying the pairing computation and the pairing function. It takes Type A pairing parameters and generates pairings. Pairing parameters could be stored in a file called *a.properties*. Figure 5.2 shows how pairings are generated. Approximately fifty percent time can be saved by preprocessing if a value is known to be paired several times. Table 5.1 shows that use of preprocessing can save almost half of the pairing generation time. Pairing parameters can be generated using the code from Figure 5.3.

```
import it.unisa.dia.gas.plaf.jpbc.pairing.a.TypeACurveGenerator;
import it.unisa.dia.gas.plaf.jpbc.pbc.curve.PBCTypeACurveGenerator;

int rBits = 160;
int qBits = 512;

// JPBC Type A pairing generator...
ParametersGenerator pg = new TypeACurveGenerator(rBits, qBits);
```

Figure 5.3: Pairing parameter generator

Setup The setup algorithm takes type A pairing parameters and generates pairings. Based on the newly generated pairings it generates a group $\mathbb{G}_1$ of prime order p with a generator g and a bilinear map

$$\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2 \tag{5.1}$$

After that the algorithm generates random elements, $t_j \in \mathbb{Z}_p$ for each attribute, $a_j \in \Omega$ from the system attribute set Ω. Let $y = \hat{e}(g, g)^{\alpha}$, where α is chosen at random from

$\mathbb{Z}_p$, and $\{T_j = g^{t_j}\}_{j=1}^{n}$. Now, the public key is published as:

$$PK = (g, y, \{T_j\}_{j=1}^{n}) \tag{5.2}$$

and the master secret key is published as:

$$MK = (\alpha, \{t_j\}_{j=1}^{n}) \tag{5.3}$$

```
Cpabe cpabe = new Cpabe();


/*setup method initializes the system by generating and storing public key and
master key in pubfile and mskfile, respectively. It also takes all
authorized attributes as input.*/
cpabe.setup(pubfile, mskfile, universe_of_authorized_attributes);
```

Figure 5.4: Setup algorithm for CP-ABE library

Key generation The secret key is generated based on a user's authorized set of attributes, S. Two parts of the secret key is generated from the key generation algorithm. One goes to the user and another to the revocation server. To avoid collusion attack each secret key gets a random value, $u_{id} \in \mathbb{Z}_p$, generated from user's unique ID, I_u. For each attribute $a_j \in S$ from the user's attribute set S a random component $u_j \in \mathbb{Z}_p$ will be generated. Library method *keygen* in Figure 5.5 generates the secret keys for the users.

```
/*keygen method is responsible for generating users' secret keys based
on their attribute set. one part of the secret key goes to the user
and other part to the revocation server*/
cpabe.keygen(pubfile, mskfile, secret_key_part1, secret_key_part2, user_attribute_set);
```

Figure 5.5: Key generation for CP-ABE library

Encryption This algorithm takes public key parameters along with plaintext file and an access policy as an input, and outputs the encrypted file, which will be stored in the server. Figure 5.6 describes the *enc* method for encryption process.

```
/*enc method takes plain data as inputfile, encrypts it with an access
policy and stores it in a file as encrypted data*/
cpabe.enc(pubfile, access_policy, inputfile, encryptedfile);
```

Figure 5.6: Encryption algorithm for CP-ABE library

Revocation check This is the first step of decryption process and revocation check is done here. When a user requests for the decryption of a file, it goes to the revocation server and if the access policy is satisfied then the server checks for the revoked attributes. A decryption token is generated if and only if no attribute from user's attribute set S is in the Attribute Revocation List (ARL). Decryption token is sent to user for decrypting the encrypted file.

```
/*m_dec_with_revocation_check method requests for decryption token
from revocation server*/
cpabe.m_dec_with_revocation_check(pubfile, secret_key_part1,
encryptedfile, decryption_token_file, user_id);
```

Figure 5.7: Generates decrpyt token with revocation check

Decryption This algorithm takes decryption token along with encrypted file and user's secret key. Once again secret key is checked against access policy of the encrypted file. If access policy is satisfied then final decryption process takes place for that specific encrypted PHR file.

```
/*after getting valid decryption token from revocation server dec
method decrypts plaintext data*/
cpabe.dec(pubfile, prvfile1part2, encfile, mdecfile, decfile,
user_id);
```

Figure 5.8: Decryption algorithm

5.2.2 Web services for the framework

We have implemented web services for different functionalities of the system. However, all the communications with the system should be encrypted using the TLS protocol. Here we list up the core web services:

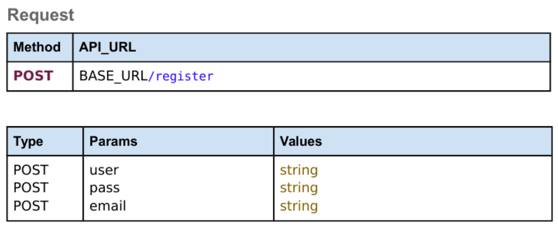
- Registration
- Login

- Get_Public_Key
- Get_Secret_Key
- Secure_Upload
- Secure_Access
- Delete_A_User
- Update_An_Account

There are other web services which will help any health organization to implement PHR system on their own premises. Using these web services we have also developed an android client application for encrypting, decrypting, and storing PHR data into the cloud. The UML diagrams of the system design are listed in Appendix A.

we describe some of our web services here :

- Registration: Using this interface user registers herself into the system.

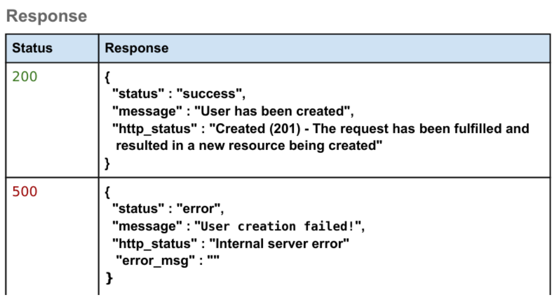


Request

Method	API_URL
POST	BASE_URL/register

Type	Params	Values
POST	user	string
POST	pass	string
POST	email	string

Figure 5.9: Request format of registration web service


Response

Status	Response
200	{ "status" : "success", "message" : "User has been created", "http_status" : "Created (201) - The request has been fulfilled and resulted in a new resource being created" }
500	{ "status" : "error", "message" : "User creation failed!", "http_status" : "Internal server error" "error_msg" : "" }

Figure 5.10: Response format of registration web service

5.2.3 Database design

PHRs are composed of several data records, which are identified by a unique positive integer number. All data records have an author, which is the user who added the record to the database. However, the record also has an owner represented by the person to whom the data refers (e.g. patients). The actual contents of the health record are encrypted and stored in the form of a blob, i.e. a sequence of bytes. The following figures describe the database design of the PHR system.

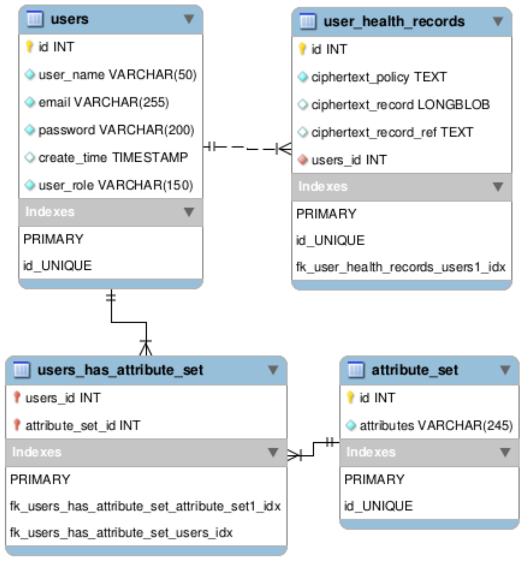
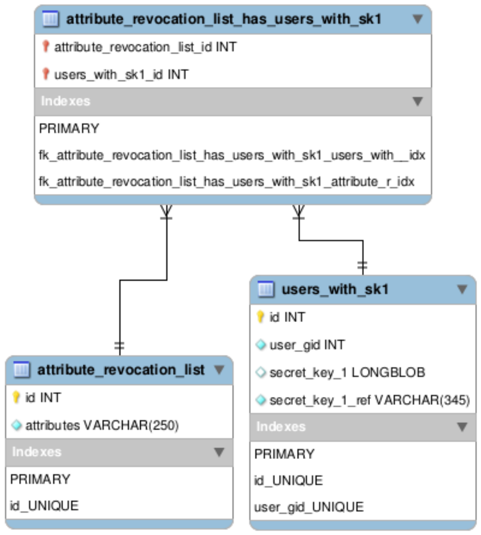


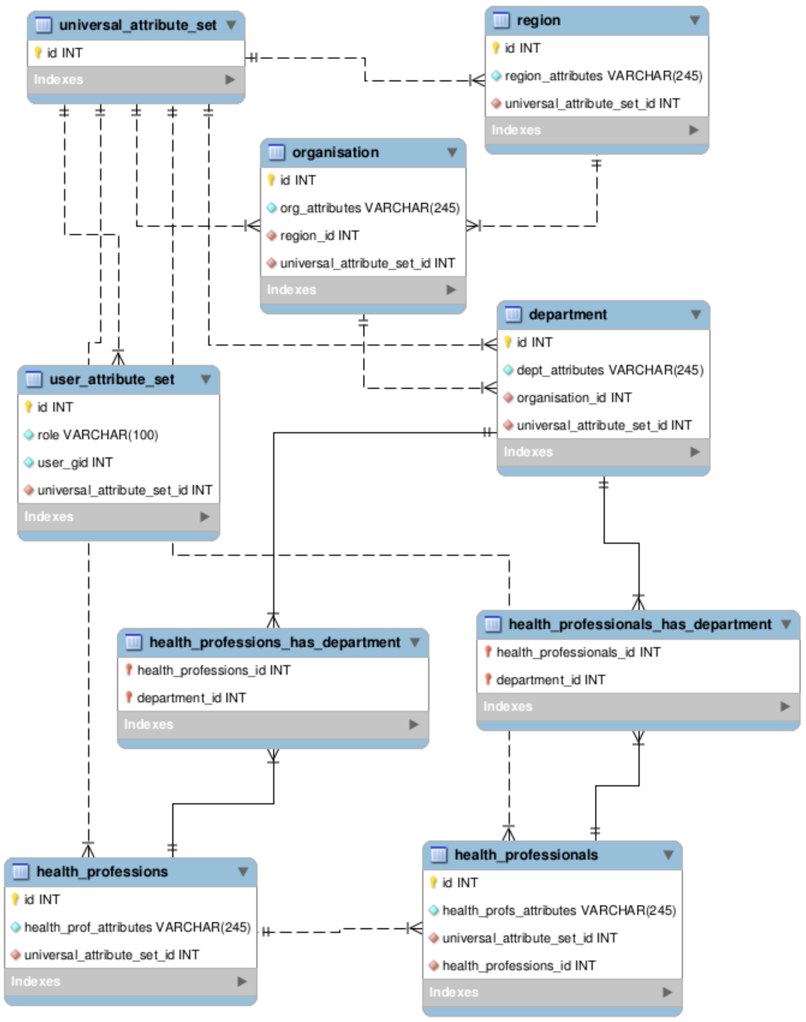
Figure 5.11: A database schema for users credentials



attribute_revocation_list_has_users_with_sk1
attribute_revocation_list_id INT
users_with_sk1_id INT
Indexes
PRIMARY
fk_attribute_revocation_list_has_users_with_sk1_users_with__idx
fk_attribute_revocation_list_has_users_with_sk1_attribute_r_idx
users_with_sk1
id INT
user_gid INT
secret_key_1 LONGBLOB
secret_key_1_ref VARCHAR(345)
Indexes
PRIMARY
id_UNIQUE
user_gid_UNIQUE
attribute_revocation_list
id INT
attributes VARCHAR(250)
Indexes
PRIMARY
id_UNIQUE

Figure 5.12: A database schema for revocation server

Hibernate framework Hibernate facilitates the storage and retrieval of Java objects. It defines a proprietary query language, Hibernate Query Language (HQL), that prevents SQL injection. Our System employs hibernate framework to execute all kind of operations with database system.



universal_attribute_set
id INT
Indexes
region
id INT
region_attributes VARCHAR(245)
universal_attribute_set_id INT
Indexes
organisation
id INT
org_attributes VARCHAR(245)
region_id INT
universal_attribute_set_id INT
Indexes
department
id INT
dept_attributes VARCHAR(245)
organisation_id INT
universal_attribute_set_id INT
Indexes
user_attribute_set
id INT
role VARCHAR(100)
user_gid INT
universal_attribute_set_id INT
Indexes
health_professions_has_department
health_professions_id INT
department_id INT
Indexes
health_professionals_has_department
health_professionals_id INT
department_id INT
Indexes
health_professions
id INT
health_prof_attributes VARCHAR(245)
universal_attribute_set_id INT
Indexes
health_professionals
id INT
health_profs_attributes VARCHAR(245)
universal_attribute_set_id INT
health_professions_id INT
Indexes

Figure 5.13: A database schema for TA server

5.2.4 Android application (PHR Client)

Here we describe the functionality of our client application. Application starts with a pop-up window to identify the user as a patient or health professional (Figure 5.14).

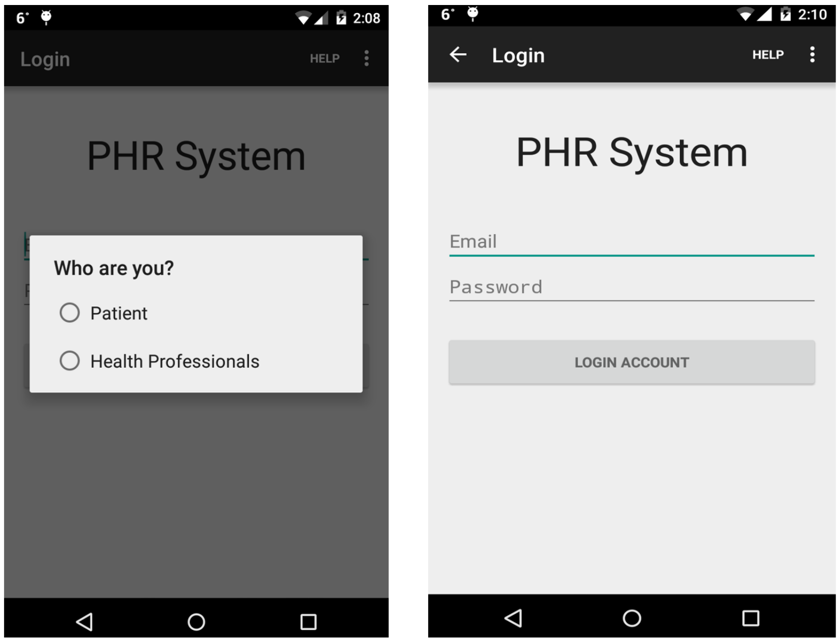
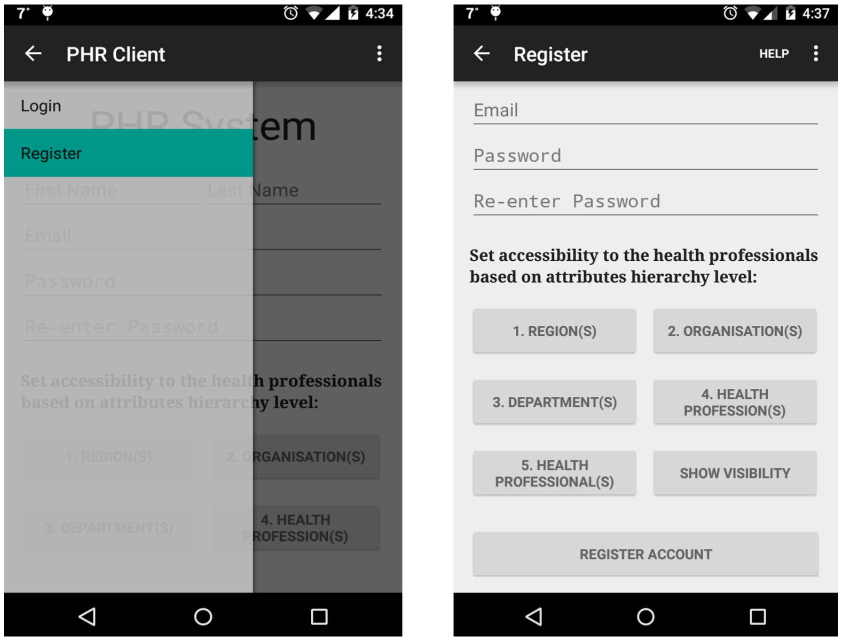


Figure 5.14: Client application with login screen

If the user tries to login as a patient using a health professional's user credentials, she would be denied access to the system. This user type-based selection process brings the users with their designated feature-based interface. We describe the client application in three different steps, user registration and login, secure upload, and secure access.

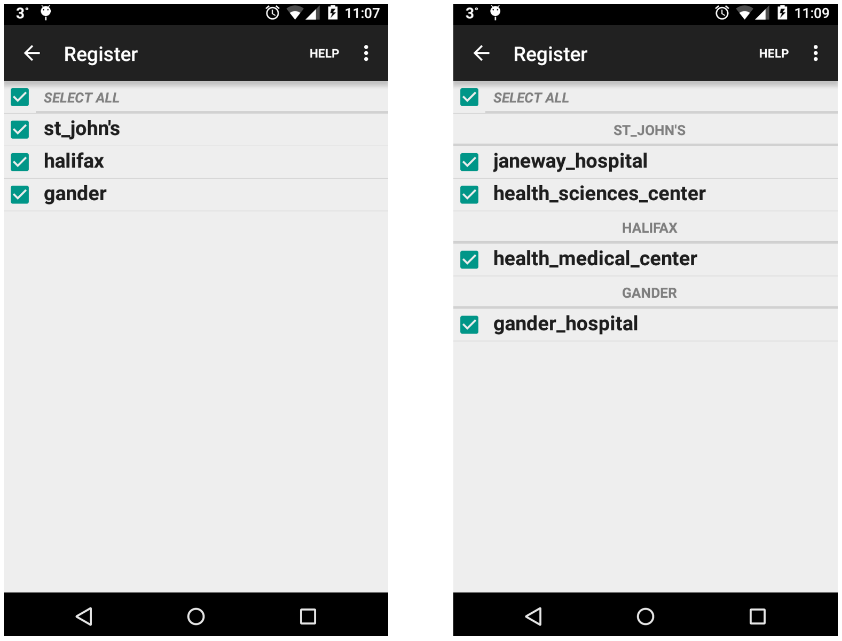
User registration and login In this interface user will register herself to use the system (Figure 5.15). Only patients will be able to register to the system. Health professionals will not be able to register themselves using this PHR Client. They will be registered to the system by the system administrator and later will be provided with their access credentials.



Set accessibility to the health professionals based on attributes hierarchy level:

Figure 5.15: Registration form

When patients register themselves, they can indicate the users who can see them and their files based on attribute hierarchy list. This accessibility option will allow only those health professionals who are authorized to access their designated patient's files. This authorization prevents health professionals to see all other unrelated patient's PHR files. Figures 5.16, 5.17 and 5.18 describe the steps of registration process. User login is fairly straightforward interface to login to the PHR system (Figure 5.15).



Register HELP

- [✓] SELECT ALL
- [✓] st_john's
- [✓] halifax
- [✓] gander

Register HELP

- [✓] SELECT ALL

ST_JOHN'S

- [✓] janeway_hospital
- [✓] health_sciences_center

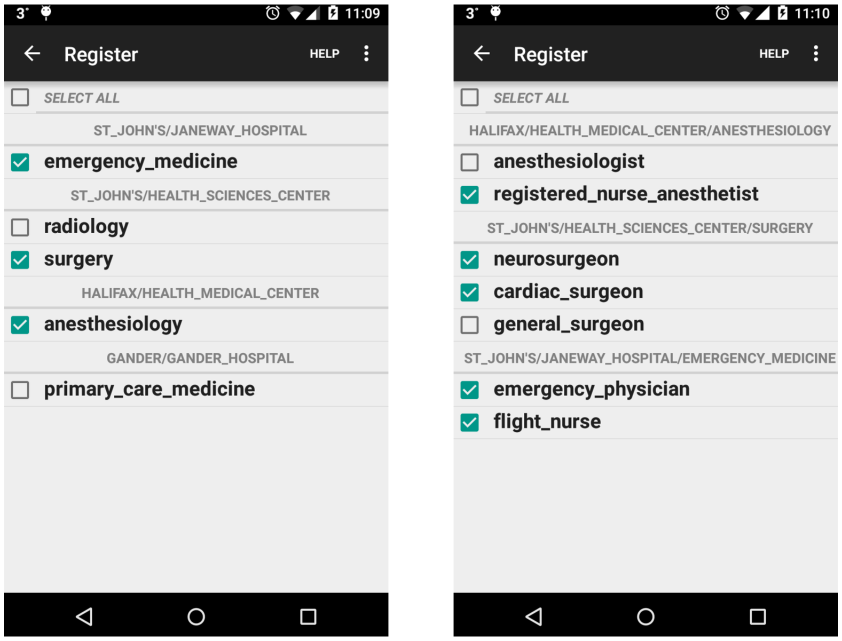
HALIFAX

- [✓] health_medical_center

GANDER

- [✓] gander_hospital

Figure 5.16: Selecting regions and respective organizations



Register HELP

- [ ] SELECT ALL

ST_JOHN'S/JANEWAY_HOSPITAL

- [✓] emergency_medicine

ST_JOHN'S/HEALTH_SCIENCES_CENTER

- [ ] radiology
- [✓] surgery

HALIFAX/HEALTH_MEDICAL_CENTER

- [✓] anesthesiology

GANDER/GANDER_HOSPITAL

- [ ] primary_care_medicine

Register HELP

- [ ] SELECT ALL

HALIFAX/HEALTH_MEDICAL_CENTER/ANESTHESIOLOGY

- [ ] anesthesiologist
- [✓] registered_nurse_anesthetist

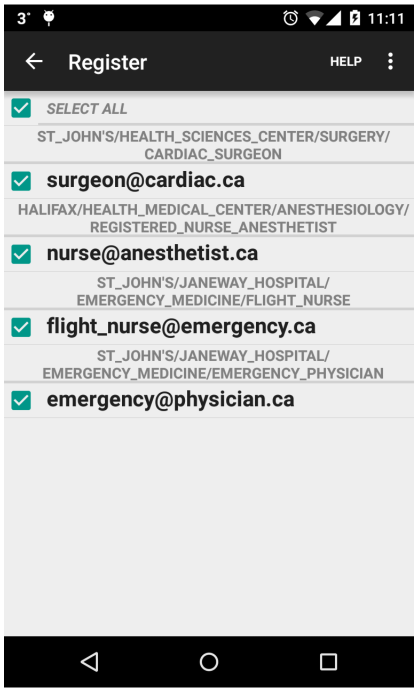
ST_JOHN'S/HEALTH_SCIENCES_CENTER/SURGERY

- [✓] neurosurgeon
- [✓] cardiac_surgeon
- [ ] general_surgeon

ST_JOHN'S/JANEWAY_HOSPITAL/EMERGENCY_MEDICINE

- [✓] emergency_physician
- [✓] flight_nurse

Figure 5.17: Selecting departments and health professions



ST_JOHN'S/HEALTH_SCIENCES_CENTER/SURGERY/CARDIAC_SURGEON
✓ surgeon@cardiac.ca
HALIFAX/HEALTH_MEDICAL_CENTER/ANESTHESIOLOGY/REGISTERED_NURSE_ANESTHETIST
✓ nurse@anesthetist.ca
ST_JOHN'S/JANEWAY_HOSPITAL/EMERGENCY_MEDICINE/FLIGHT_NURSE
✓ flight_nurse@emergency.ca
ST_JOHN'S/JANEWAY_HOSPITAL/EMERGENCY_MEDICINE/EMERGENCY_PHYSICIAN
✓ emergency@physician.ca

Figure 5.18: Selecting health professionals

Secure upload User selects a file from her device to encrypt. Then she defines an access policy based on system attributes. “Secure Upload” button performs two steps of operation. First, encrypts the file with defined access policy and then uploads it to the server (Figure 5.19).

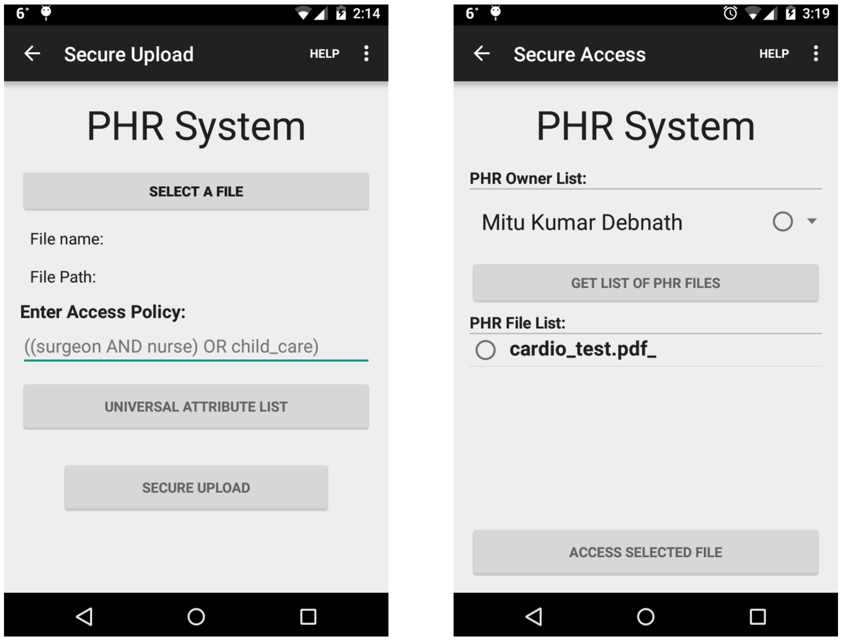
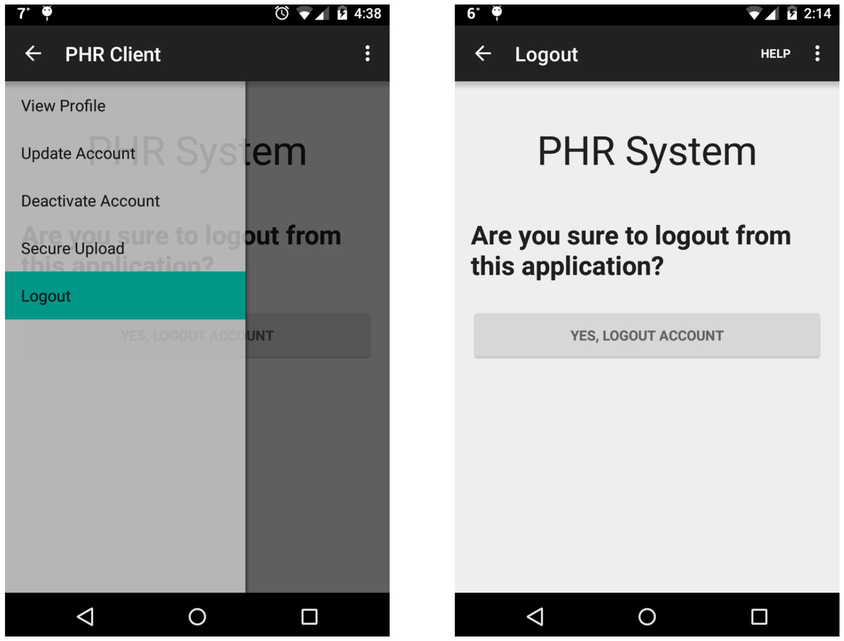


Figure 5.19: Secure upload and secure access

Secure access A health professional gets the list of patients and selects a particular patient from the list. Another list of PHR files is generated based on the selected user. Then health professional can select whichever file she wants to access and press "Access Selected File" button. This also does two step operations. First it gets the encrypted file and then decrypts it. After that file is stored to the file system of the device from where user can access decrypted file (Figure 5.19).

Figure 5.20 describes user logout screen to terminate current user session.



PHR System

Are you sure to logout from this application?

Figure 5.20: User logout screen

Chapter 6

Experimental Results And Analysis

6.1 Experimental results

We run some test cases on our CP-ABE library of the PHR system. We also use dummy PHR data to run the test in the system. We used 65 attributes as system attributes. Public key and master key are created based on those attributes.

```
/** The attribute set 1. */
      static String attr1 = "doc@cardiologist.ca nurse@mentalhealth.ca anesthesiologist
          nurse_anesthetist emergency_physician_assistant";

/** The attribute set 2. */
      static String attr2 = "cardiovascular_medicine pediatric_medicine primary_care_medicine
          neurologist";
```

Figure 6.1: Two different attribute sets

```
/** The access policy. */
        static String policy = "((cardiovascular_medicine AND pediatric_medicine) OR
              (primary_care_medicine AND doc@cardiologist.ca))";
```

Figure 6.2: Access policy

A set of secret keys are generated based on two sets of user attributes (Figure 6.1). An access policy (Figure 6.2) is used to encrypt PHR data files. Our test has been conducted under three scenarios:

- We encrypt PHR file with the access policy from Figure 6.2 and try to decrypt it using the user's secret keys created from attribute set 1 (Figure 6.1). It must be noted that whenever we talk about secret keys we actually identify two parts of the user secret key. From the access policy we can see that only those secret keys can decrypt the PHR file which have "cardiovascular_medicine" and "pediatric_medicine" or "primary_care_medicine" and "doc@cardiologist.ca" in their attribute set. Secret keys created from attribute set 1 do not satisfy this access policy. The reason is that first two attributes from access policy do not exist in set 1 and only one attribute from second two attributes exists in set 1 which clearly does not satisfy the access policy either. That's why we received AttributesNotSatisfiedException (Figure 6.3) while trying to decrypt.

```
//cannot decrypt, attributes in key do not satisfy policy.
Exception in thread "main"
com.mitu.utils.exceptions.AttributesNotSatisfiedException: Attributes
are not satisfied.
```

Figure 6.3: Access policy is not satisfied

- Next, we tried to decrypt the PHR file using secret keys created from attribute set 2 (Figure 6.1). The secret keys clearly satisfy the access policy because both "cardiovascular_medicine" and "pediatric_medicine" attributes are in the secret keys. So the PHR file is decrypted successfully.

- In the last scenario, we also tried to decrypt the PHR file using secret keys based on attribute set 2. But this time we added "cardiovascular_medicine" and "primary_care_medicine" in the Attribute Revocation List (ARL) from attribute set 2 and revocation server denied to serve with an exception called NoSuchDecryptionTokenFoundException (Figure 6.4). Server generates this type of exception when attributes are revoked and no decryption token is generated.

```
//Attribute has been revoked!!
Exception in thread "main"
com.mitu.utils.exceptions.NoSuchDecryptionTokenFoundException:
Attributes have been revoked.
```

Figure 6.4: Attributes are revoked

6.2 Performance analysis

In this section, we provide the experimental results to evaluate the performance and feasibility of our proposed PHR system. The experiment is performed on:

- 64-bit Linux workstation with AMD Athlon(tm) Dual-core 2.1 GHz Processor and 3GB RAM.
- Google Nexus 4 with Android OS 5.1.1 and Qualcomm Snapdragon S4 Quad-core 1500 MHz Processor.

Our CP-ABE library implementation is based on Type A pairings, which uses a 160-bit elliptic curve group on the curve $y^2 = x^3 + x$ over a 512-bit finite field [10]. Our scheme is based on the Java porting of Pairing-Based Cryptography (jPBC) library version 2.0.0. All the experimental results are mostly the average of 4/5 trials to obtain a consistent output because CPU clock time may provide different results based on other processes running on the current machine. All the run time includes system time as well.

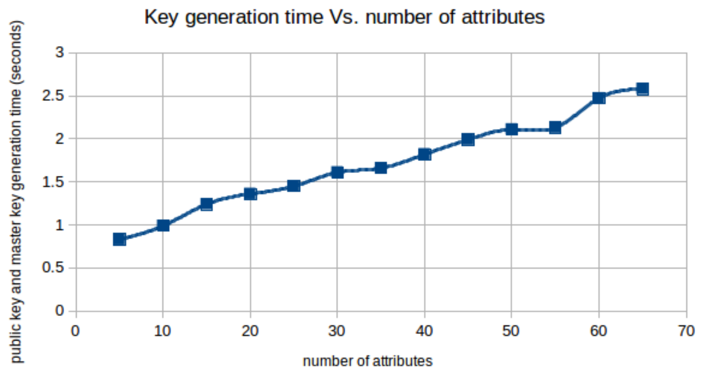


Figure 6.5: Setup time

We analyze overall performance of the system based on CP-ABE library. We performed encryption and decryption in the same user device for the analysis purpose. Practical testing can be done with different user devices. Figure 6.5 describes the public key and master key generation time based on different number of attributes. From this graph we can see that key generation time increases almost linearly with number of attributes in the system. Key generation time with 45 attributes takes 1.99 seconds, which is only 1 second more than the key generation time with 10 attributes.

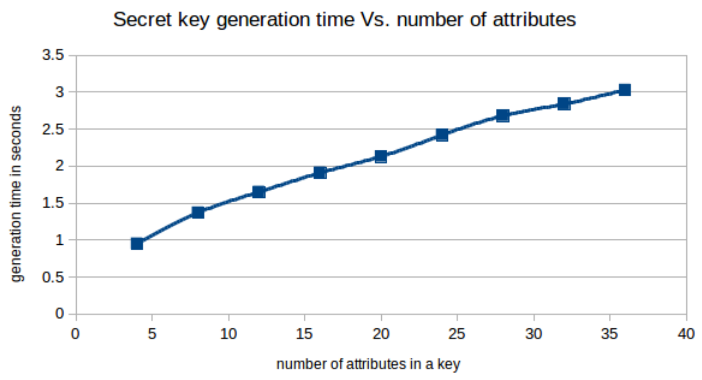


Figure 6.6: Secret key generation time

User secret key generation doesn't depend on the entire system attribute set. It only depends on the user attribute set. In our system, secret keys are generated in two parts and both of them take almost same time to be created. Figure 6.6 describes secret key generation time based on different sets of user attributes. The time also increases linearly as the number of attributes increases.

We encrypt six different PHR files (1.2MB, 3.8MB, 8.2MB, 12.2MB, 16.7MB, 20MB) with eight different numbers of attributes in the access policy to measure performance of the encryption operations. Access policy complexity plays important role in the speed of encryption operations. Here the speed of encryption (Figure 6.7) decreases, while the size of the access policy increases and this is almost linear with the number of attributes in the access policy.

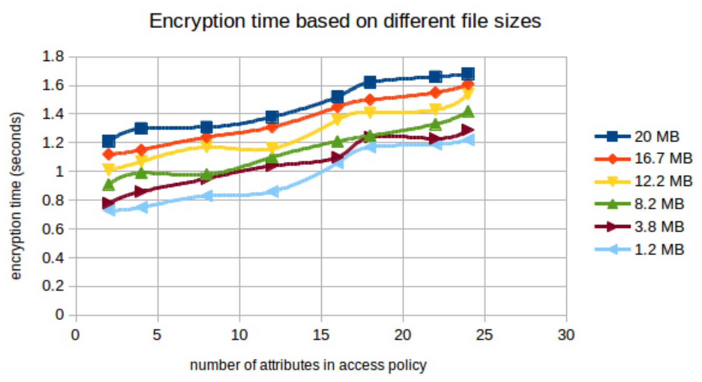


Figure 6.7: Encryption time based on different file sizes

However, decryption time (Figure 6.8) tends to be almost constant regardless of different attribute sizes in access policy as we are decrypting files using a single secret key.

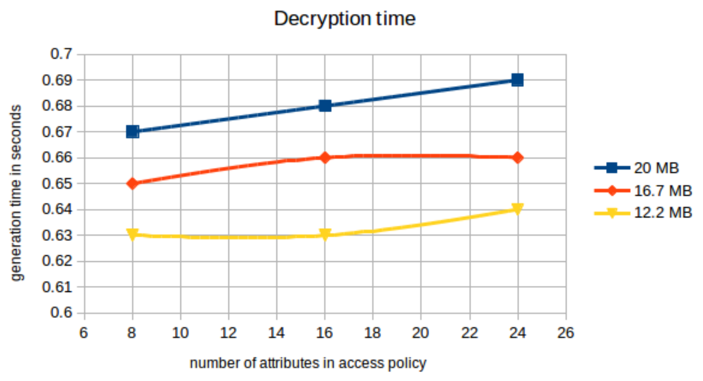


Figure 6.8: Decryption time based on different file sizes

During the attribute revocation, revocation server only needs to check Attribute Revocation List (ARL) for the revoked attributes in the access policy, which doesn't add any extra overhead for decryption token generation time. From Figure 6.9 we can see that decryption token generation is the same as decryption operation.

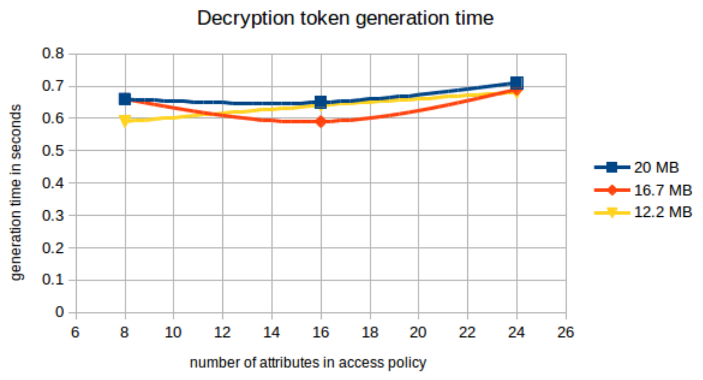


Figure 6.9: Decryption token generation time in revocation server

In our system, PHR files around 1MB are encrypted in less than 0.8 seconds depending on the number of attributes in the access policy. Files ranging from 1MB to 20MB take time from 0.7 to 1.7 seconds to encrypt, as shown in Figure 6.7. On the other hand, decryption time (Figure 6.8) for files ranging from 12MB to 20MB are from 0.63 to 0.69 seconds. It is obvious that lower size files will take less than 0.6 seconds to decrypt. These results indicate that the time for these operations is feasible for PHR systems.

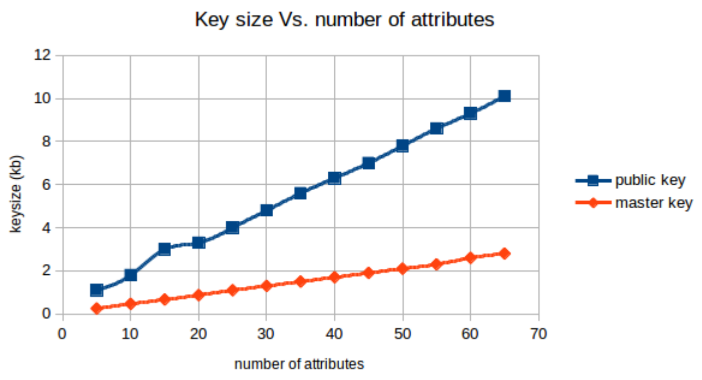


Figure 6.10: Public and master key sizes

From Figure 6.10, the size of public key increases linearly with the number of attributes. But master key size doesn't increase like public key. Public key size increases up to 10KB for 65 attributes whereas master key size grows only to around 3KB. Secret key size also increases linearly with the number of user attributes. Key size for 4 attributes is around 700 bytes, and for 36 attributes is 5.5KB. These key sizes are negligible in terms of storage overhead.

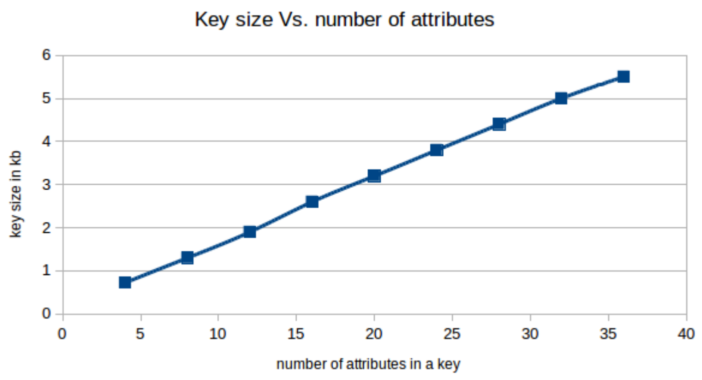


Figure 6.11: Secret key size based on number of attributes

The size of the ciphertext depends on the size of the access policy. we measured the difference between ciphertext and plaintext across various numbers of attributes in the access policy. The difference is almost identical for all the files. The difference is around 5KB for access policy with 22 attributes and around 4KB for access policy with 18 attributes.

Chapter 7

Conclusions And Future Work

7.1 Summary

PHR system is an essential part of any healthcare organization, because PHR consists of all the medical information of a particular patient. PHR system helps users to get best and yet faster treatments for their health problems. At the same time, users also want to keep their sensitive PHR data secured so that only authorized personnel can get access to them.

In this thesis, we proposed a framework to share PHR data effectively and securely. We also implemented this framework for sharing personal health records in a secured way by using a variation of generic CP-ABE encryption scheme named mCP-ABE [23]. To accomplish our goal we implemented the Java library of this CP-ABE scheme with the help of jPBC library, which is the first implementation of this kind to our best knowledge. This framework would allow users to share, manage, and process PHR data with fine grained access control mechanism. In this framework, patients have full control on their PHR data. Moreover, this framework efficiently handles the fundamental challenge of key management, which is introduced by multiple PHR users and owners.

Furthermore, it handles efficient and on-demand attribute revocation. Revocation can happen due to the following reasons: i) User attributes are invalid, or ii) User attributes have been compromised. Our framework handles this revocation process effectively. Revocation server maintains an Attribute Revocation List (ARL) for this purpose. When an attribute is revoked from the system attribute list, Ω, TA removes it and notifies revocation server to put it in revocation list right away, which instantiates instant revocation. TA can revoke system and user attributes, but attribute owner can also put request to revoke attribute if she thinks it is compromised.

7.2 Future work

As a future direction of this work system attribute set can be made flexible by supporting new attribute definitions after the initialization of the system. MySQL RDBMS can be replaced by NoSQL database system (e.g., MongoDB, Oracle NoSQL Database, etc.). NoSQL database is suitable for PHR applications, because it is easier to manage and it provides a higher level of flexibility. Also, to make the system more robust, and to overcome the issue of single point of failure, single Trusted Authority (TA) can be replaced by a (k, n) threshold scheme [29], such that at least k TAs should participate to generate the user secret keys.

Bibliography

[1] Ben Adida, Arjun Sanyal, Steve Zabak, Isaac S. Kohane, and Kenneth D. Mandl. Indivo X: developing a fully substitutable personally controlled health record platform. In *AMIA Annual Symposium Proceedings*, volume 2010, page 6. American Medical Informatics Association, 2010.

[2] Joseph A. Akinyele, Christoph U. Lehmann, Matthew D. Green, Matthew W. Pagano, Zachary NJ Peterson, and Aviel D. Rubin. Self-protecting electronic medical records using attribute-based encryption. 2010.

[3] Mrinmoy Barua, Xiaohui Liang, Rongxing Lu, and Xuemin Shen. Peace: An efficient and secure patient-centric access control scheme for ehealth care system. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 970–975. IEEE, 2011.

[4] Josh Benaloh, Melissa Chase, Eric Horvitz, and Kristin Lauter. Patient controlled encryption: ensuring privacy of electronic medical records. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 103–114. ACM, 2009.

[5] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*,

pages 321–334. IEEE, 2007.

[6] Ian F. Blake, Gadiel Seroussi, and Nigel Smart. *Elliptic curves in cryptography*, volume 265. Cambridge university press, 1999.

[7] CareCloud. Healthcare and medical billing software, http://www.carecloud.com/, 2015.

[8] Ling Cheung, Joseph A. Cooley, Roger Khazan, and Calvin Newport. Collusion-resistant group key management using attribute-based encryption. *Group-Oriented Cryptographic Protocols*, page 23, 2007.

[9] John Craig and Victor Patterson. Introduction to the practice of telemedicine. *Journal of Telemedicine and Telecare*, 11(1):3–9, 2005.

[10] Angelo De Caro and Vincenzo Iovino. jPBC: Java pairing based cryptography. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pages 850–855. IEEE, 2011.

[11] Tim Dierks. The transport layer security (TLS) protocol (version 1.2), 2008.

[12] Judy Foreman. At risk of exposure : In the push for electronic medical records, concern is growing about how well privacy can be safeguarded. *Los Angeles Times*, June 2006.

[13] Markle Foundation. The common framework: Overview and principles. Connecting for health, 2006.

[14] Practice Fusion. Free electronic health records, http://www.practicefusion.com/, 2015.

[15] Martin Gagne. Applications of bilinear maps in cryptography. 2002.

[16] Steven D. Galbraith, Keith Harrison, and David Soldera. Implementing the tate pairing. In *Algorithmic number theory*, pages 324–337. Springer, 2002.

[17] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[18] Paul Garrett. Intro abstract algebra, 1997.

[19] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm, 2006.

[20] Trisha Greenhalgh, Susan Hinder, Katja Stramer, Tanja Bratan, and Jill Russell. Adoption, non-adoption, and abandonment of a personal electronic health record: case study of HealthSpace. *Bmj*, 341, 2010.

[21] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[22] Microsoft HealthVault. http://www.healthvault.com, 2015.

[23] Luan Ibraimi, Milan Petkovic, Svetla Nikova, Pieter Hartel, and Willem Jonker. Mediated ciphertext-policy attribute-based encryption and its application. In *Information Security Applications*, pages 309–323. Springer, 2009.

[24] Cisco inc. Internet of everything. http://ioeassessment.cisco.com/, 2015.

[25] Antoine Joux. A one round protocol for tripartite Diffie–Hellman. In *Algorithmic number theory*, pages 385–393. Springer, 2000.

[26] David C. Kaelber, Ashish K. Jha, Douglas Johnston, Blackford Middleton, and David W. Bates. A research agenda for personal health records (PHRs). *Journal of the American Medical Informatics Association*, 15(6):729–736, 2008.

[27] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[28] Neal Koblitz, Alfred Menezes, and Scott Vanstone. The state of elliptic curve cryptography. In *Towards a Quarter-Century of Public Key Cryptography*, pages 103–123. Springer, 2000.

[29] Jun Kurihara, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka. A new (k, n)-threshold secret sharing scheme and its extension. In *Information Security*, pages 455–470. Springer, 2008.

[30] Ben Lynn. On the implementation of pairing-based cryptosystems. Stanford University, 2007.

[31] Ben Lynn. *The pairing-based cryptography (PBC) library.* 2010.

[32] Kenneth D. Mandl, Peter Szolovits, Isaac S. Kohane, David Markwell, Rhona MacDonald, and others. Public standards and patients' control: how to keep electronic medical records accessible but private. *Bmj*, 322(7281):283–287, 2001.

[33] Alfred Menezes. An introduction to pairing-based cryptography. volume 477, pages 47–65, 2005.

[34] Victor S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology—CRYPTO'85 Proceedings*, pages 417–426. Springer, 1986.

[35] David Niewolny. How the internet of things is revolutionizing healthcare. *White paper, Oct*, 2013.

[36] iHealthBeat org. Google, microsoft say HIPAA stimulus rule doesn't apply to them, April 2009.

[37] iHealthBeat org. Hackers directly targeting health care organizations, FBI warns, August 2014.

[38] Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Brent Waters. Secure attribute-based systems. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 99–112. ACM, 2006.

[39] Ronald L. Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[40] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Advances in Cryptology–EUROCRYPT 2005*, pages 457–473. Springer, 2005.

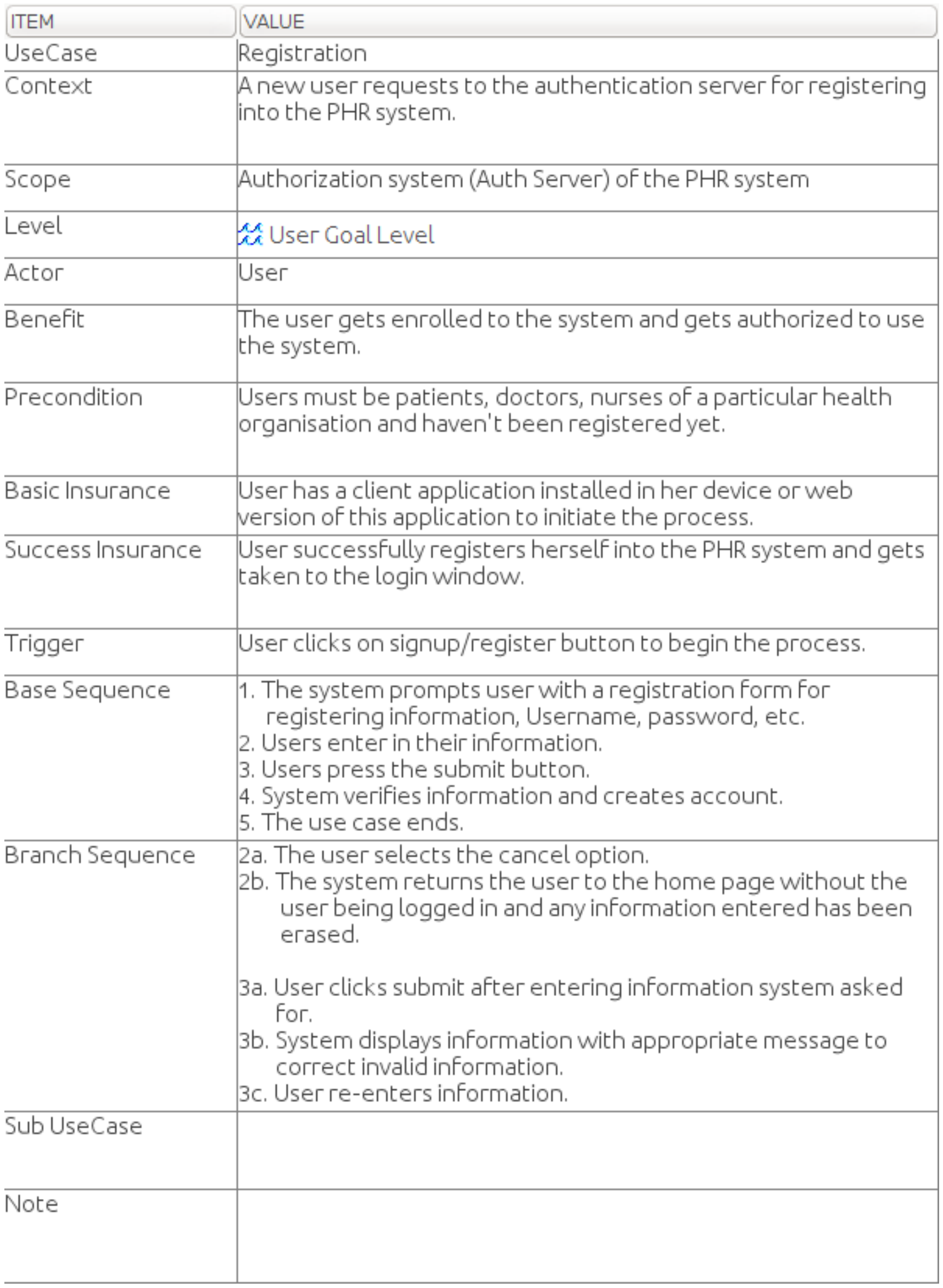
[41] William Stallings. The advanced encryption standard. *Cryptologia*, 26(3):165–188, 2002.

[42] Scott A. Vanstone. Next generation security for wireless: elliptic curve cryptography. *Computers & Security*, 22(5):412–415, 2003.

Appendix A

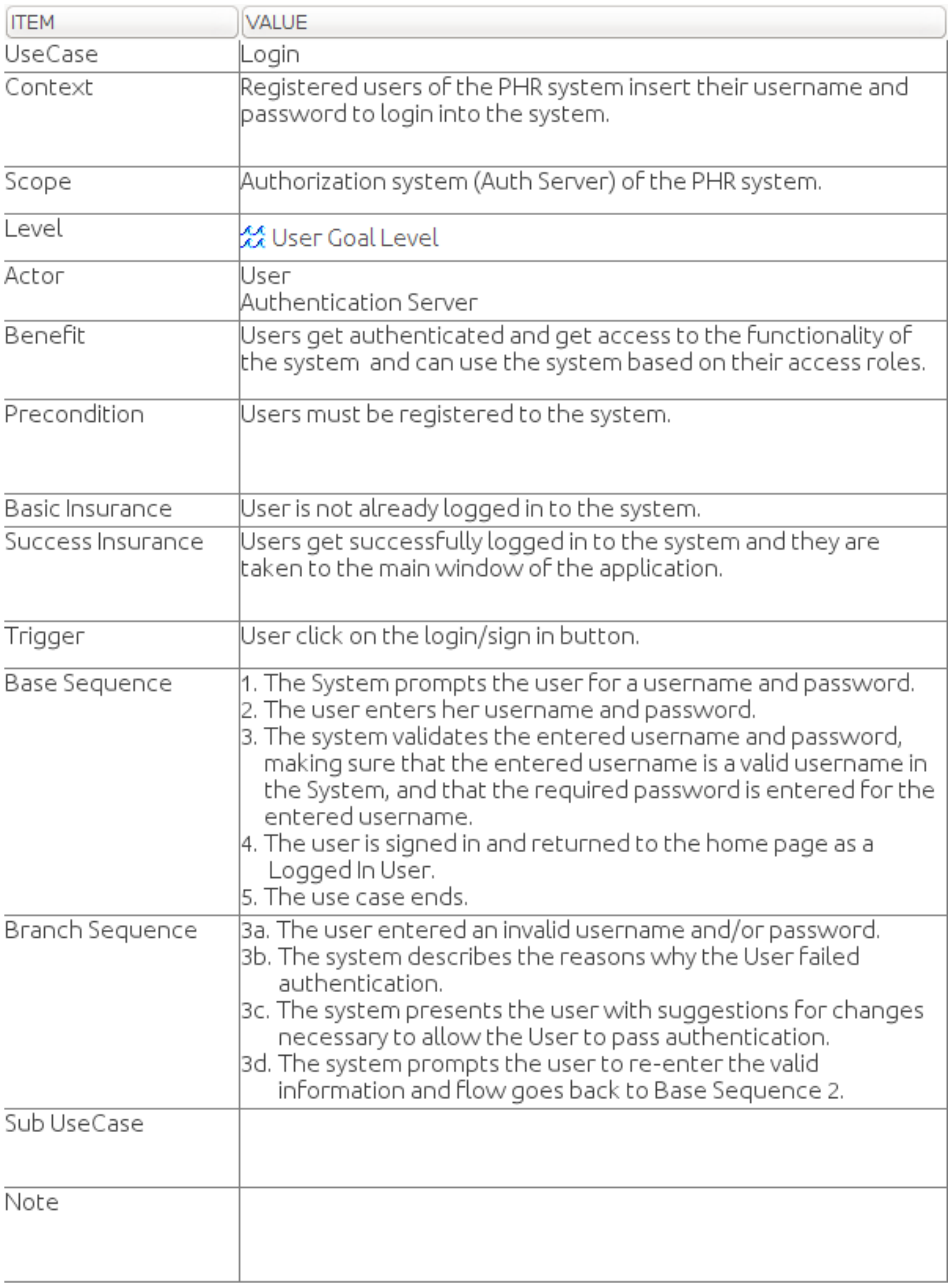
UML Diagrams

Here we list up all the use case descriptions.



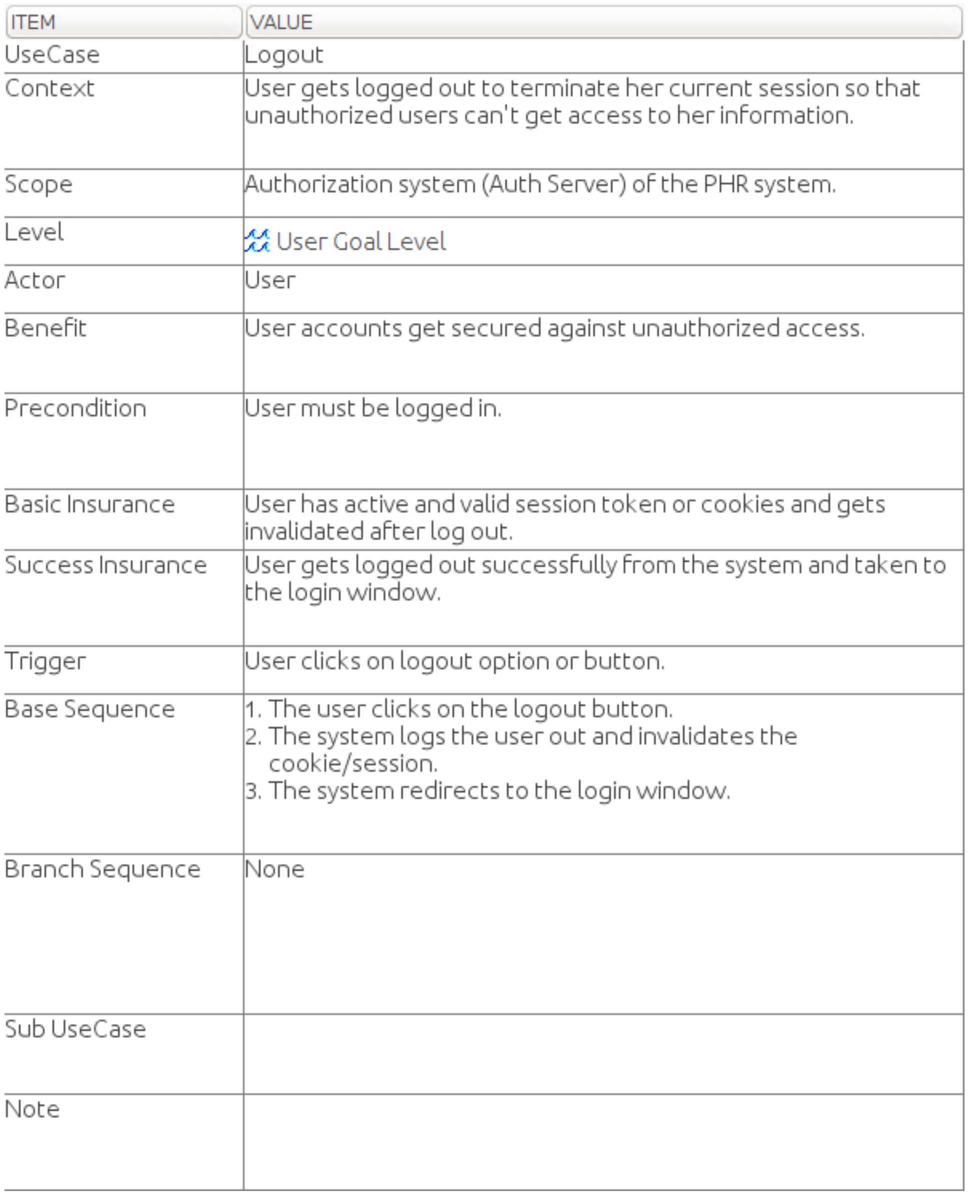
ITEM	VALUE
UseCase	Registration
Context	A new user requests to the authentication server for registering into the PHR system.
Scope	Authorization system (Auth Server) of the PHR system
Level	User Goal Level
Actor	User
Benefit	The user gets enrolled to the system and gets authorized to use the system.
Precondition	Users must be patients, doctors, nurses of a particular health organisation and haven't been registered yet.
Basic Insurance	User has a client application installed in her device or web version of this application to initiate the process.
Success Insurance	User successfully registers herself into the PHR system and gets taken to the login window.
Trigger	User clicks on signup/register button to begin the process.
Base Sequence	1. The system prompts user with a registration form for registering information, Username, password, etc. 2. Users enter in their information. 3. Users press the submit button. 4. System verifies information and creates account. 5. The use case ends.
Branch Sequence	2a. The user selects the cancel option. 2b. The system returns the user to the home page without the user being logged in and any information entered has been erased. 3a. User clicks submit after entering information system asked for. 3b. System displays information with appropriate message to correct invalid information. 3c. User re-enters information.
Sub UseCase	
Note	

Figure A.1: Use case description: Registration



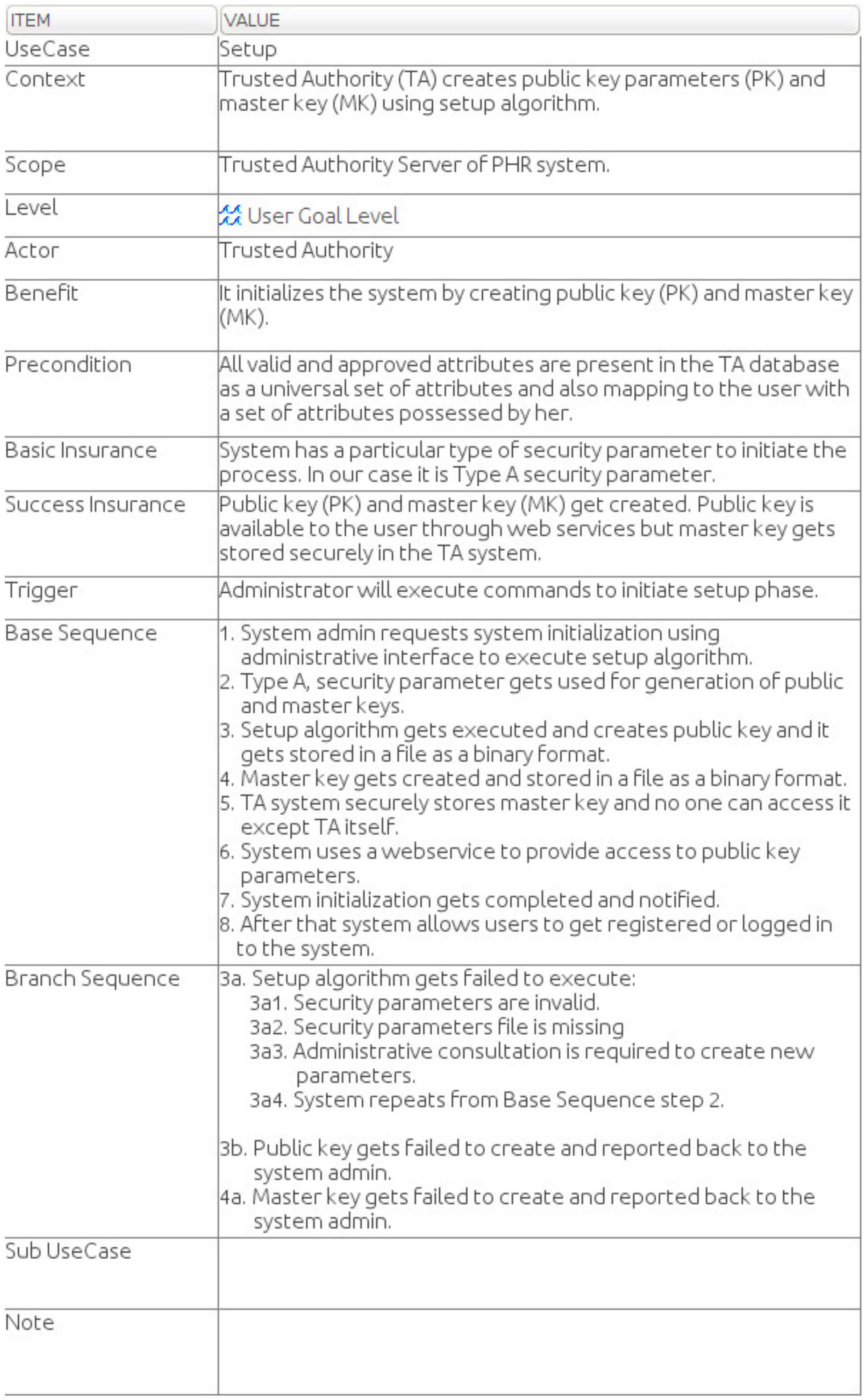
ITEM	VALUE
UseCase	Login
Context	Registered users of the PHR system insert their username and password to login into the system.
Scope	Authorization system (Auth Server) of the PHR system.
Level	User Goal Level
Actor	User Authentication Server
Benefit	Users get authenticated and get access to the functionality of the system and can use the system based on their access roles.
Precondition	Users must be registered to the system.
Basic Insurance	User is not already logged in to the system.
Success Insurance	Users get successfully logged in to the system and they are taken to the main window of the application.
Trigger	User click on the login/sign in button.
Base Sequence	1. The System prompts the user for a username and password. 2. The user enters her username and password. 3. The system validates the entered username and password, making sure that the entered username is a valid username in the System, and that the required password is entered for the entered username. 4. The user is signed in and returned to the home page as a Logged In User. 5. The use case ends.
Branch Sequence	3a. The user entered an invalid username and/or password. 3b. The system describes the reasons why the User failed authentication. 3c. The system presents the user with suggestions for changes necessary to allow the User to pass authentication. 3d. The system prompts the user to re-enter the valid information and flow goes back to Base Sequence 2.
Sub UseCase	
Note	

Figure A.2: Use case description: Login



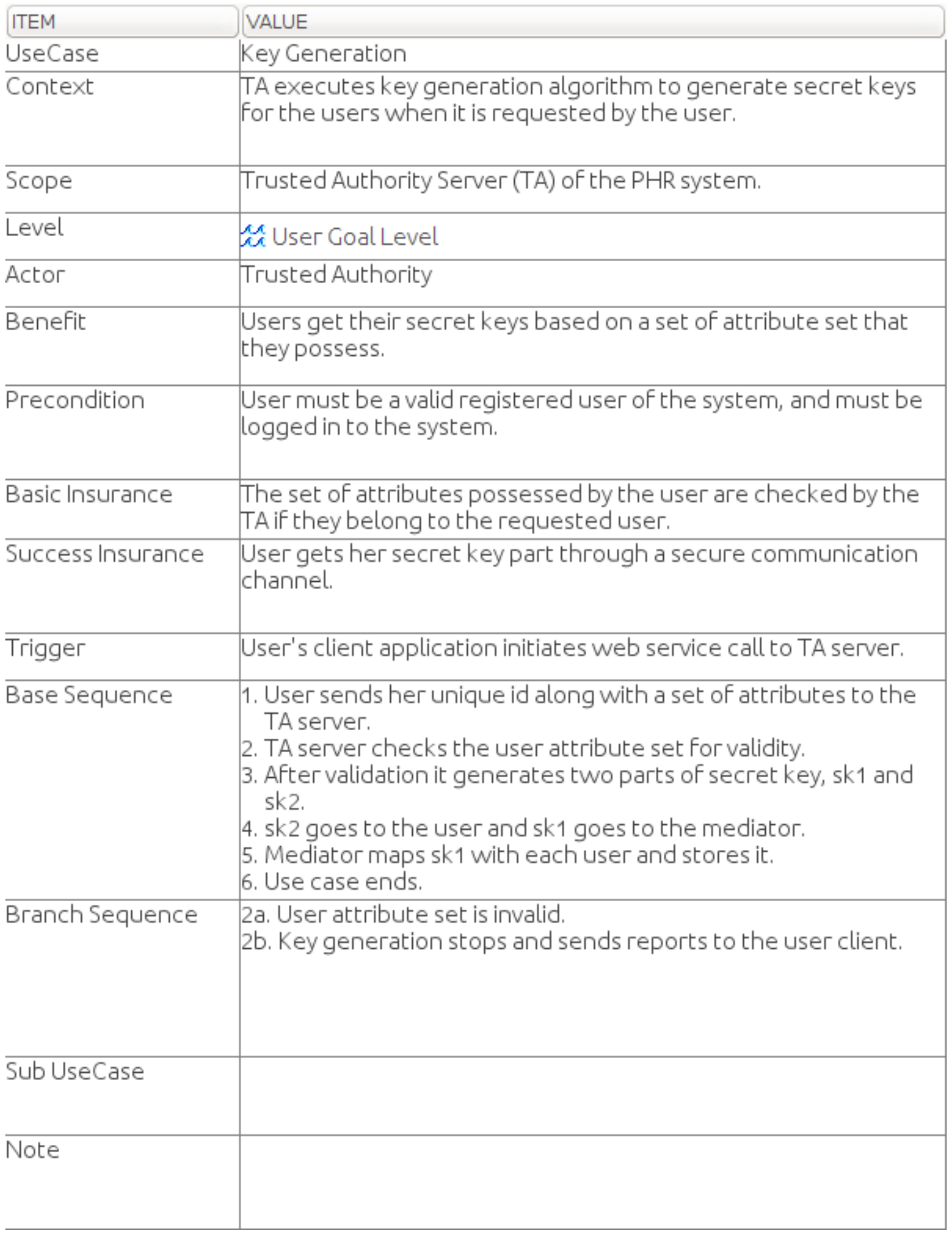
ITEM	VALUE
UseCase	Logout
Context	User gets logged out to terminate her current session so that unauthorized users can't get access to her information.
Scope	Authorization system (Auth Server) of the PHR system.
Level	User Goal Level
Actor	User
Benefit	User accounts get secured against unauthorized access.
Precondition	User must be logged in.
Basic Insurance	User has active and valid session token or cookies and gets invalidated after log out.
Success Insurance	User gets logged out successfully from the system and taken to the login window.
Trigger	User clicks on logout option or button.
Base Sequence	1. The user clicks on the logout button. 2. The system logs the user out and invalidates the cookie/session. 3. The system redirects to the login window.
Branch Sequence	None
Sub UseCase	
Note	

Figure A.3: Use case description: Logout



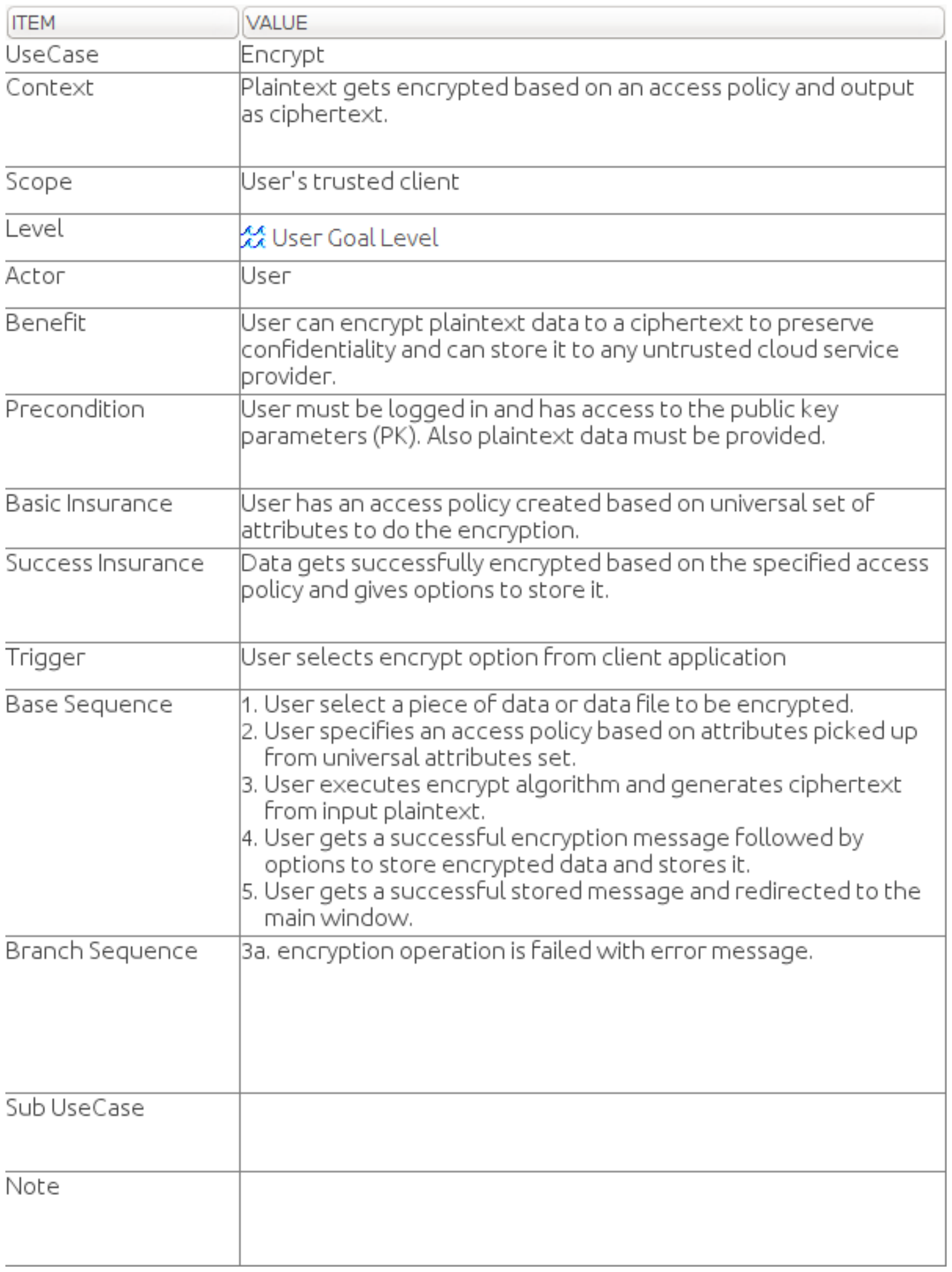
ITEM	VALUE
UseCase	Setup
Context	Trusted Authority (TA) creates public key parameters (PK) and master key (MK) using setup algorithm.
Scope	Trusted Authority Server of PHR system.
Level	User Goal Level
Actor	Trusted Authority
Benefit	It initializes the system by creating public key (PK) and master key (MK).
Precondition	All valid and approved attributes are present in the TA database as a universal set of attributes and also mapping to the user with a set of attributes possessed by her.
Basic Insurance	System has a particular type of security parameter to initiate the process. In our case it is Type A security parameter.
Success Insurance	Public key (PK) and master key (MK) get created. Public key is available to the user through web services but master key gets stored securely in the TA system.
Trigger	Administrator will execute commands to initiate setup phase.
Base Sequence	1. System admin requests system initialization using administrative interface to execute setup algorithm. 2. Type A, security parameter gets used for generation of public and master keys. 3. Setup algorithm gets executed and creates public key and it gets stored in a file as a binary format. 4. Master key gets created and stored in a file as a binary format. 5. TA system securely stores master key and no one can access it except TA itself. 6. System uses a webservice to provide access to public key parameters. 7. System initialization gets completed and notified. 8. After that system allows users to get registered or logged in to the system.
Branch Sequence	3a. Setup algorithm gets failed to execute: 3a1. Security parameters are invalid. 3a2. Security parameters file is missing 3a3. Administrative consultation is required to create new parameters. 3a4. System repeats from Base Sequence step 2. 3b. Public key gets failed to create and reported back to the system admin. 4a. Master key gets failed to create and reported back to the system admin.
Sub UseCase	
Note	

Figure A.4: Use case description: Setup



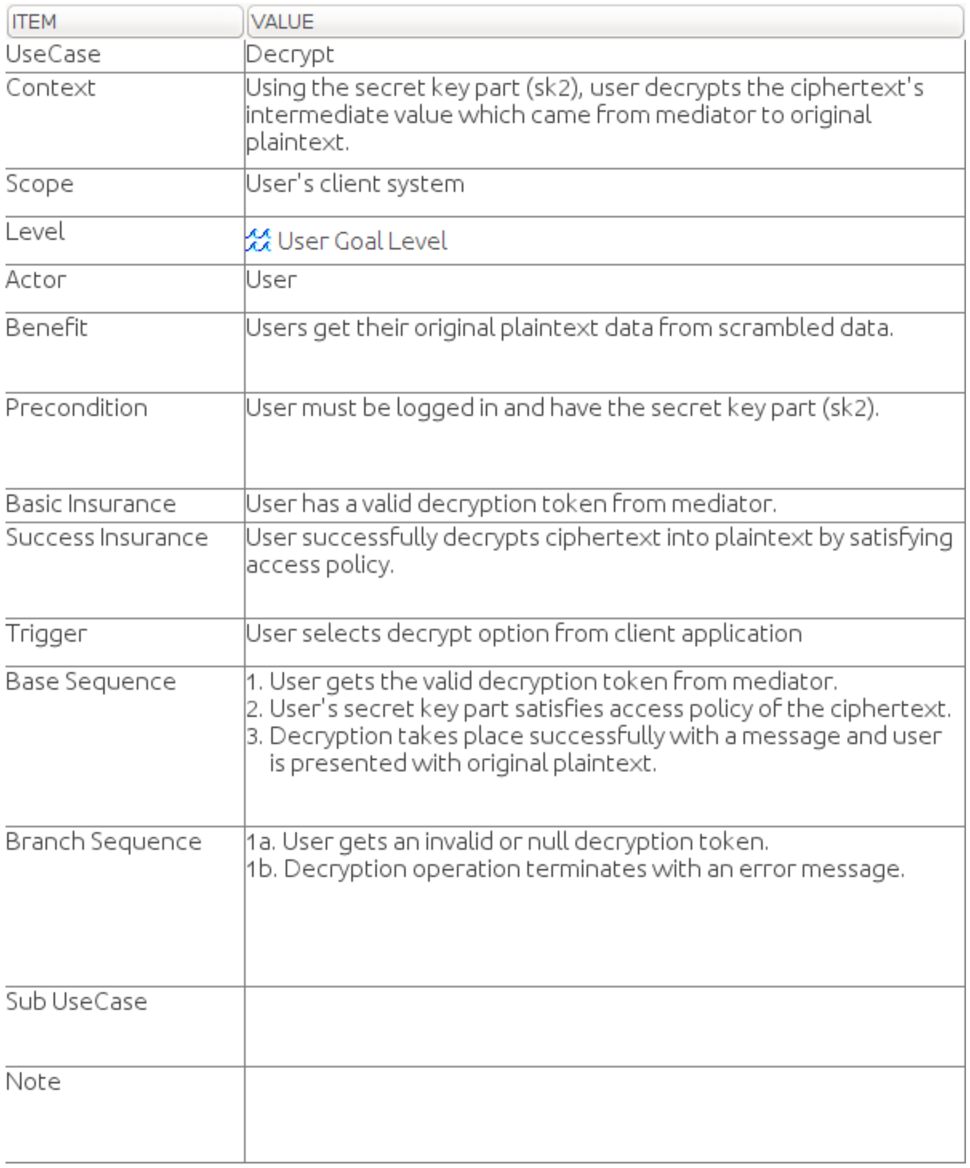
ITEM	VALUE
UseCase	Key Generation
Context	TA executes key generation algorithm to generate secret keys for the users when it is requested by the user.
Scope	Trusted Authority Server (TA) of the PHR system.
Level	User Goal Level
Actor	Trusted Authority
Benefit	Users get their secret keys based on a set of attribute set that they possess.
Precondition	User must be a valid registered user of the system, and must be logged in to the system.
Basic Insurance	The set of attributes possessed by the user are checked by the TA if they belong to the requested user.
Success Insurance	User gets her secret key part through a secure communication channel.
Trigger	User's client application initiates web service call to TA server.
Base Sequence	1. User sends her unique id along with a set of attributes to the TA server. 2. TA server checks the user attribute set for validity. 3. After validation it generates two parts of secret key, sk1 and sk2. 4. sk2 goes to the user and sk1 goes to the mediator. 5. Mediator maps sk1 with each user and stores it. 6. Use case ends.
Branch Sequence	2a. User attribute set is invalid. 2b. Key generation stops and sends reports to the user client.
Sub UseCase	
Note	

Figure A.5: Use case description: Key Generation



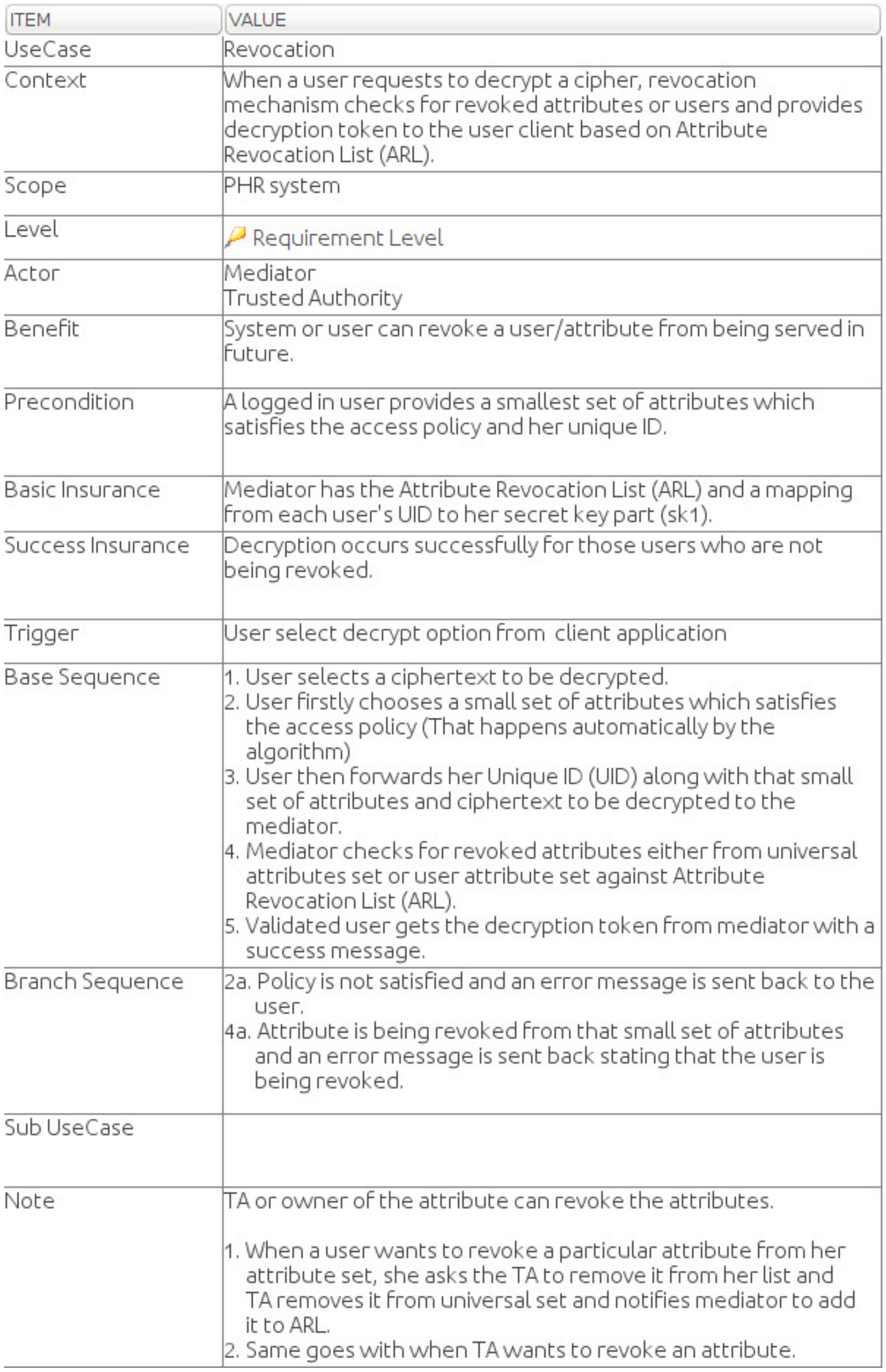
ITEM	VALUE
UseCase	Encrypt
Context	Plaintext gets encrypted based on an access policy and output as ciphertext.
Scope	User's trusted client
Level	User Goal Level
Actor	User
Benefit	User can encrypt plaintext data to a ciphertext to preserve confidentiality and can store it to any untrusted cloud service provider.
Precondition	User must be logged in and has access to the public key parameters (PK). Also plaintext data must be provided.
Basic Insurance	User has an access policy created based on universal set of attributes to do the encryption.
Success Insurance	Data gets successfully encrypted based on the specified access policy and gives options to store it.
Trigger	User selects encrypt option from client application
Base Sequence	1. User select a piece of data or data file to be encrypted. 2. User specifies an access policy based on attributes picked up from universal attributes set. 3. User executes encrypt algorithm and generates ciphertext from input plaintext. 4. User gets a successful encryption message followed by options to store encrypted data and stores it. 5. User gets a successful stored message and redirected to the main window.
Branch Sequence	3a. encryption operation is failed with error message.
Sub UseCase	
Note	

Figure A.6: Use case description: Encryption



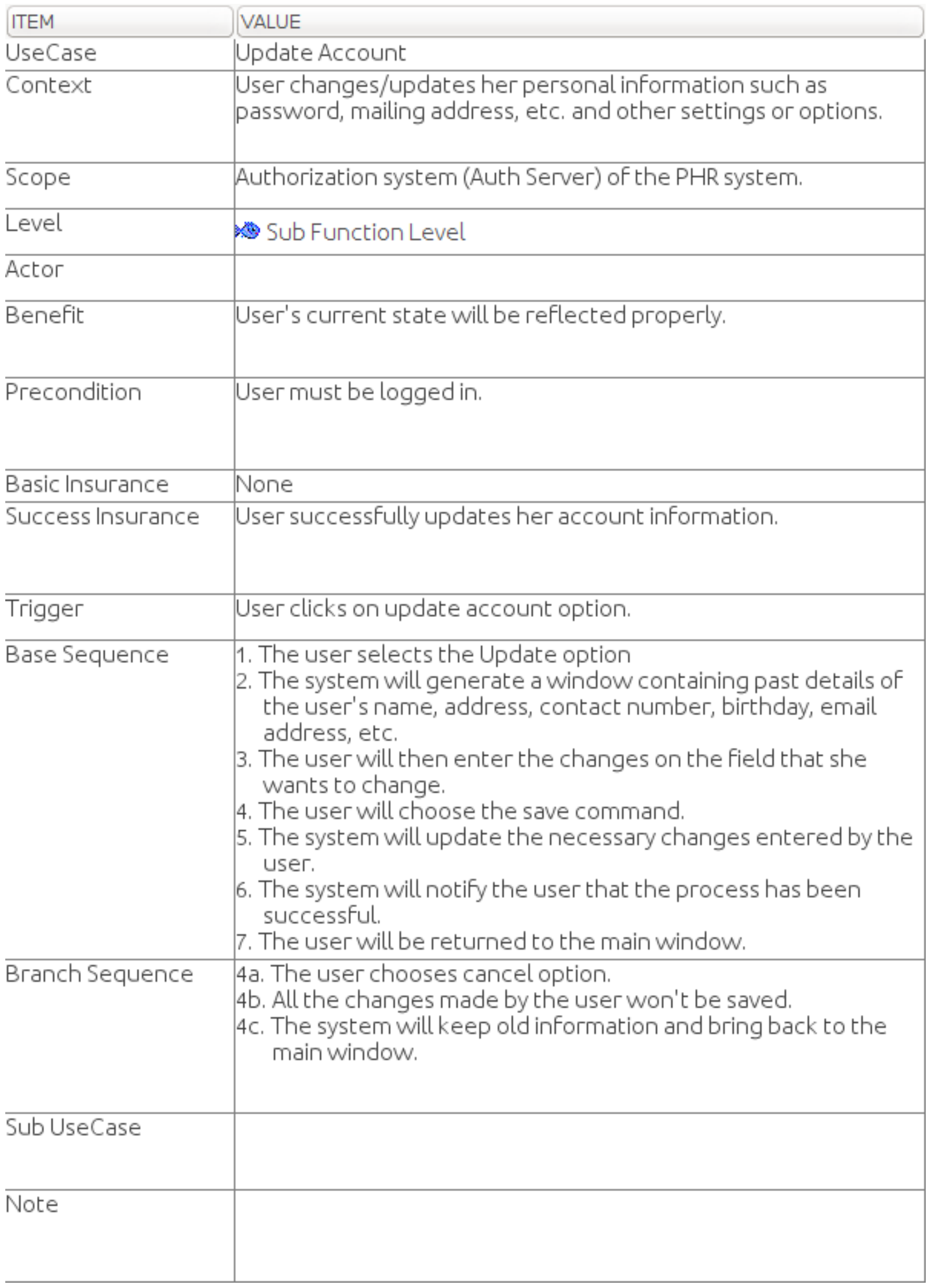
ITEM	VALUE
UseCase	Decrypt
Context	Using the secret key part (sk2), user decrypts the ciphertext's intermediate value which came from mediator to original plaintext.
Scope	User's client system
Level	User Goal Level
Actor	User
Benefit	Users get their original plaintext data from scrambled data.
Precondition	User must be logged in and have the secret key part (sk2).
Basic Insurance	User has a valid decryption token from mediator.
Success Insurance	User successfully decrypts ciphertext into plaintext by satisfying access policy.
Trigger	User selects decrypt option from client application
Base Sequence	1. User gets the valid decryption token from mediator. 2. User's secret key part satisfies access policy of the ciphertext. 3. Decryption takes place successfully with a message and user is presented with original plaintext.
Branch Sequence	1a. User gets an invalid or null decryption token. 1b. Decryption operation terminates with an error message.
Sub UseCase	
Note	

Figure A.7: Use case description: Decryption



ITEM	VALUE
UseCase	Revocation
Context	When a user requests to decrypt a cipher, revocation mechanism checks for revoked attributes or users and provides decryption token to the user client based on Attribute Revocation List (ARL).
Scope	PHR system
Level	Requirement Level
Actor	Mediator Trusted Authority
Benefit	System or user can revoke a user/attribute from being served in future.
Precondition	A logged in user provides a smallest set of attributes which satisfies the access policy and her unique ID.
Basic Insurance	Mediator has the Attribute Revocation List (ARL) and a mapping from each user's UID to her secret key part (sk1).
Success Insurance	Decryption occurs successfully for those users who are not being revoked.
Trigger	User select decrypt option from client application
Base Sequence	1. User selects a ciphertext to be decrypted. 2. User firstly chooses a small set of attributes which satisfies the access policy (That happens automatically by the algorithm) 3. User then forwards her Unique ID (UID) along with that small set of attributes and ciphertext to be decrypted to the mediator. 4. Mediator checks for revoked attributes either from universal attributes set or user attribute set against Attribute Revocation List (ARL). 5. Validated user gets the decryption token from mediator with a success message.
Branch Sequence	2a. Policy is not satisfied and an error message is sent back to the user. 4a. Attribute is being revoked from that small set of attributes and an error message is sent back stating that the user is being revoked.
Sub UseCase	
Note	TA or owner of the attribute can revoke the attributes. 1. When a user wants to revoke a particular attribute from her attribute set, she asks the TA to remove it from her list and TA removes it from universal set and notifies mediator to add it to ARL. 2. Same goes with when TA wants to revoke an attribute.

Figure A.8: Use case description: Revocation



ITEM	VALUE
UseCase	Update Account
Context	User changes/updates her personal information such as password, mailing address, etc. and other settings or options.
Scope	Authorization system (Auth Server) of the PHR system.
Level	Sub Function Level
Actor	
Benefit	User's current state will be reflected properly.
Precondition	User must be logged in.
Basic Insurance	None
Success Insurance	User successfully updates her account information.
Trigger	User clicks on update account option.
Base Sequence	1. The user selects the Update option 2. The system will generate a window containing past details of the user's name, address, contact number, birthday, email address, etc. 3. The user will then enter the changes on the field that she wants to change. 4. The user will choose the save command. 5. The system will update the necessary changes entered by the user. 6. The system will notify the user that the process has been successful. 7. The user will be returned to the main window.
Branch Sequence	4a. The user chooses cancel option. 4b. All the changes made by the user won't be saved. 4c. The system will keep old information and bring back to the main window.
Sub UseCase	
Note	

Figure A.9: Use case description: Update User Account

These are the activity diagram of the framework design:

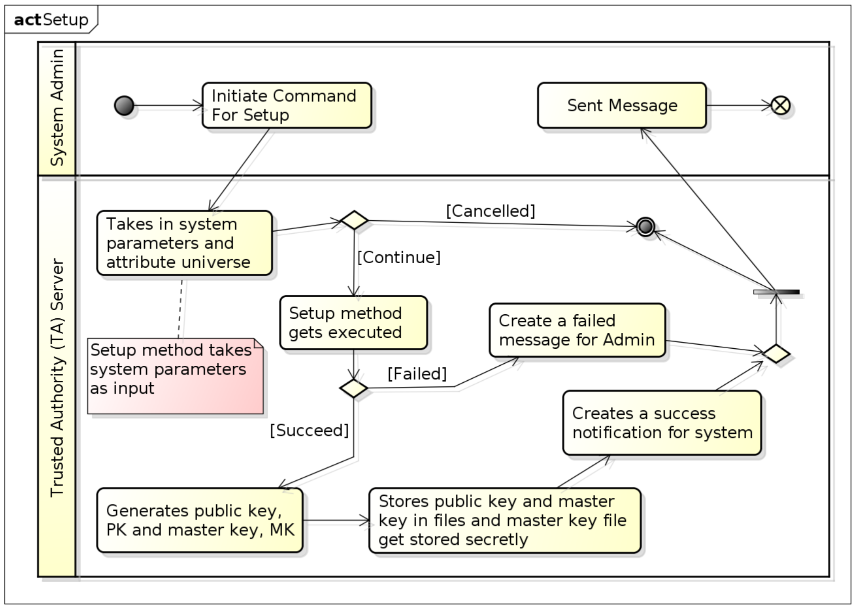

actSetup
System Admin
Trusted Authority (TA) Server
Initiate Command For Setup
Sent Message
Takes in system parameters and attribute universe
[Cancelled]
[Continue]
Setup method gets executed
Create a failed message for Admin
Setup method takes system parameters as input
[Failed]
[Succeed]
Creates a success notification for system
Generates public key, PK and master key, MK
Stores public key and master key in files and master key file get stored secretly

Figure A.10: System initialization by generating public and master keys

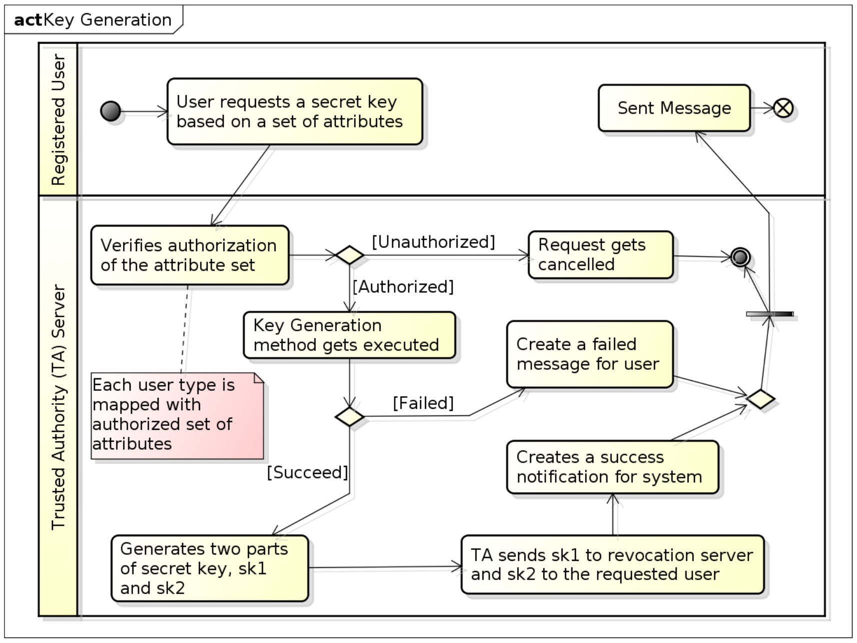

actKey Generation
Registered User
Trusted Authority (TA) Server
User requests a secret key based on a set of attributes
Sent Message
Verifies authorization of the attribute set
[Unauthorized]
Request gets cancelled
[Authorized]
Key Generation method gets executed
Create a failed message for user
Each user type is mapped with authorized set of attributes
[Failed]
Creates a success notification for system
[Succeed]
Generates two parts of secret key, sk1 and sk2
TA sends sk1 to revocation server and sk2 to the requested user

Figure A.11: Generates user secret key based on an attribute set.

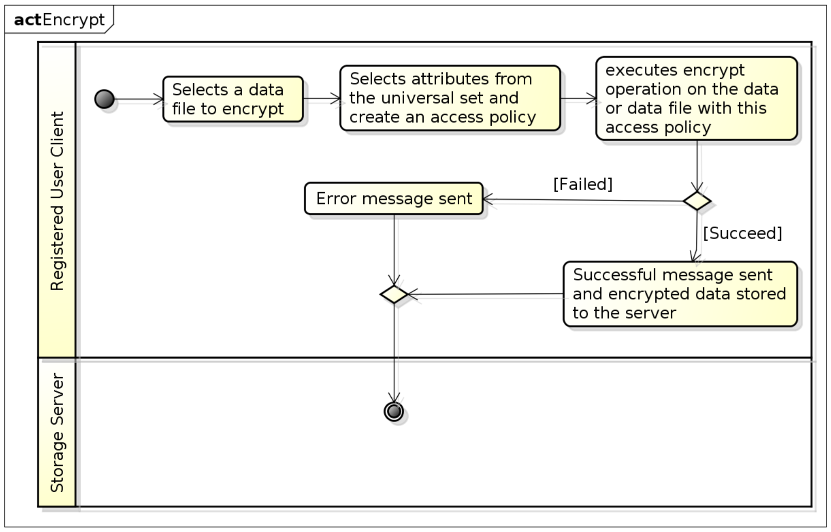

actEncrypt
Registered User Client
Storage Server
Selects a data file to encrypt
Selects attributes from the universal set and create an access policy
executes encrypt operation on the data or data file with this access policy
[Failed]
[Succeed]
Error message sent
Successful message sent and encrypted data stored to the server

Figure A.12: Encryption process of PHR data

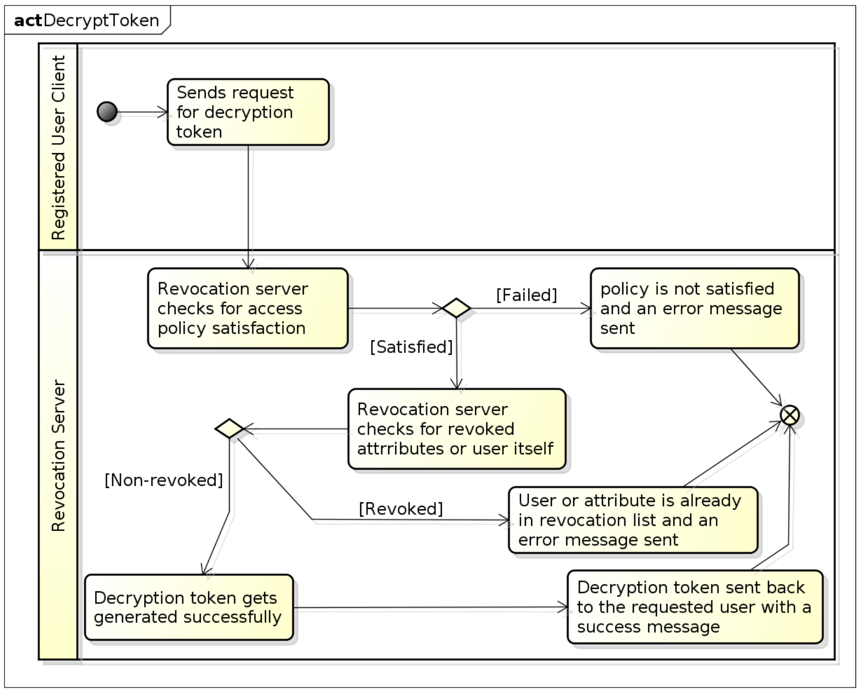

actDecryptToken
Registered User Client
Revocation Server
Sends request for decryption token
Revocation server checks for access policy satisfaction
[Failed]
[Satisfied]
policy is not satisfied and an error message sent
Revocation server checks for revoked attrributes or user itself
[Non-revoked]
[Revoked]
User or attribute is already in revocation list and an error message sent
Decryption token gets generated successfully
Decryption token sent back to the requested user with a success message

Figure A.13: Requesting decryption token

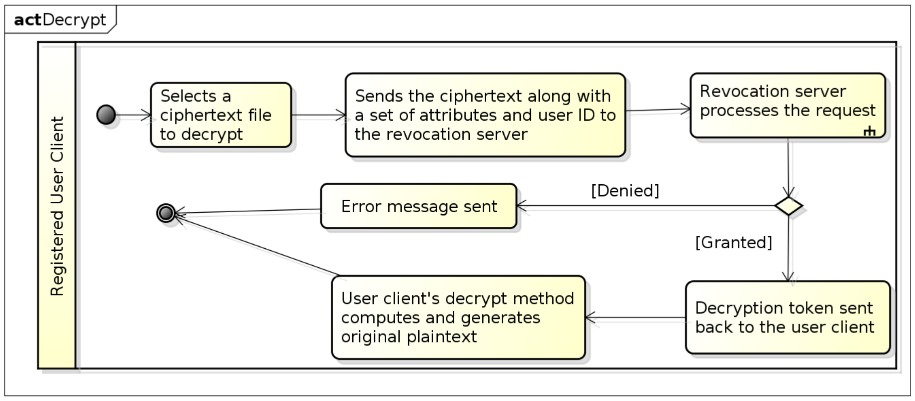

actDecrypt
Registered User Client
Selects a ciphertext file to decrypt
Sends the ciphertext along with a set of attributes and user ID to the revocation server
Revocation server processes the request
[Denied]
Error message sent
[Granted]
Decryption token sent back to the user client
User client's decrypt method computes and generates original plaintext

Figure A.14: Decryption process of PHR data

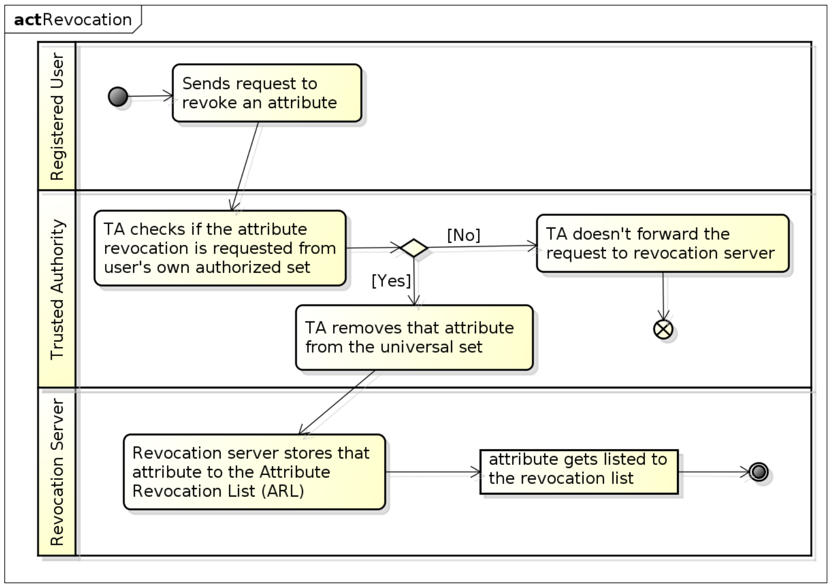

actRevocation
Registered User
Trusted Authority
Revocation Server
Sends request to revoke an attribute
TA checks if the attribute revocation is requested from user's own authorized set
[No]
TA doesn't forward the request to revocation server
[Yes]
TA removes that attribute from the universal set
Revocation server stores that attribute to the Attribute Revocation List (ARL)
attribute gets listed to the revocation list

Figure A.15: Attribute revocation based on ARL

Appendix B

Programming Languages And Tools

Technologies and programming tools, which we have used for implementation, are listed below:

- **Web Server/Application Server:** Any Java Application Server (JAS) such as, Jboss, Glassfish, etc. or any web server such as, tomcat or jetty can be used. We used Apache Tomcat 8.
- **Database Server:** We have used MySQL Server 5.6 for database system.
- **Editor (IDE):** Both netbeans and eclipse IDE can be used for web service development. Specifically, we used Eclipse Mars for web service development and ADT for android client development.
- **Programming Language:** We used JAVA for web service development (server side). For client application development we used Android Platform (client side).

- **Web Service Framework:** We used RESTlet 2.2.2 framework for designing RESTful web services.

- **Persistence Framework:** We used Hibernate 4.3 framework for persisting data to backend database system (In our case, it is MySQL DBMS).

- **Project Build:** For maintaining project structure and building automatically along with dependencies we used Maven 2, a project building tool.

Appendix C

CP-ABE Library : JAVA Source Code

This is the Cpabe.java class file of our CP-ABE library. Setup, key generation, encryption, and decryption operations are invoked from this class. We present only this main class of our library here.

```java
package com.mitu.cpabe;


import it.unisa.dia.gas.jpbc.Element;
import java.io.IOException;
import java.security.NoSuchAlgorithmException;
import java.util.logging.Level;
import java.util.logging.Logger;
import com.mitu.abe.Abe;
import com.mitu.abe.AbeCph;
import com.mitu.abe.AbeCphKey;
import com.mitu.abe.AbeMDec;
import com.mitu.abe.AbeMsk;
import com.mitu.abe.AbePrv;
import com.mitu.abe.AbePrvPart1;
import com.mitu.abe.AbePrvPart2;

import com.mitu.abe.AbePub;

import com.mitu.abe.SerializeUtils;

import com.mitu.cpabe.policy.LangPolicy;

import com.mitu.utils.exceptions.AttributesNotSatisfiedException;

import com.mitu.utils.exceptions.NoSuchDecryptionTokenFoundException;


// TODO: Auto-generated Javadoc

/**

 * The Class Cpabe.

 */

public class Cpabe {


       /**

        * Setup.

        *

        * @author Mitu Kumar Debnath

        * @param pubfile

        *           the pubfile

        * @param mskfile

        *           the mskfile

        * @param attrs

        *           the attrs

        * @throws IOException

        *            Signals that an I/O exception has occurred.

        * @throws ClassNotFoundException

        *            the class not found exception

        */


       public void setup(String pubfile, String mskfile, String[] attrs) throws IOException {


              byte[] pub_byte, msk_byte;


              AbePub pub = new AbePub();

              AbeMsk msk = new AbeMsk();

              Abe.setup(pub, msk, attrs);
        /* store AbePub into pubfile */
        pub_byte = SerializeUtils.serializeBswabePub(pub);
        Common.spitFile(pubfile, pub_byte);

        /* store AbeMsk into mskfile */
        msk_byte = SerializeUtils.serializeBswabeMsk(msk);
        Common.spitFile(mskfile, msk_byte);

}

/**
 * Keygen.
 *
 * @param pubfile
 *            the pubfile
 * @param prvfilePart1
 *            the prvfile part1
 * @param prvfilePart2
 *            the prvfile part2
 * @param mskfile
 *            the mskfile
 * @param attr_str
 *            the attr_str
 * @throws NoSuchAlgorithmException
 *             the no such algorithm exception
 * @throws IOException
 *             Signals that an I/O exception has occurred.
 * @throws ClassNotFoundException
 */
public void keygen(String pubfile, String prvfilePart1, String prvfilePart2, String mskfile,
     String attr_str)
                throws NoSuchAlgorithmException, IOException {

        AbePub pub;
        AbeMsk msk;
        byte[] pub_byte, msk_byte, prv_bytePart1, prv_bytePart2;

        /* get AbePub from pubfile */
        pub_byte = Common.suckFile(pubfile);
        pub = SerializeUtils.unserializeBswabePub(pub_byte);

        // ObjectInputStream inputStream = Utility.deSerializeObject(pubfile);
        // pub = (AbePub) inputStream.readObject();

        /* get AbeMsk from mskfile */
        msk_byte = Common.suckFile(mskfile);
        msk = SerializeUtils.unserializeBswabeMsk(pub, msk_byte);

        // ObjectInputStream inputStream2 = Utility.deSerializeObject(mskfile);
        // msk = (AbeMsk) inputStream2.readObject();

        String[] attr_arr = null;
        try {
                attr_arr = LangPolicy.parseAttribute(attr_str);
        } catch (Exception ex) {
                Logger.getLogger(Cpabe.class.getName()).log(Level.SEVERE, null, ex);
        }
        AbePrv prv = Abe.keygen(pub, msk, attr_arr);

        /* store AbePrv into prvfile */
        prv_bytePart1 = SerializeUtils.serializeBswabePrvPart1(prv.prv1);
        prv_bytePart2 = SerializeUtils.serializeBswabePrvPart2(prv.prv2);
        Common.spitFile(prvfilePart1, prv_bytePart1);
        Common.spitFile(prvfilePart2, prv_bytePart2);

        // Utility.serializeObject(prvfilePart1, prv.prv1);
        // Utility.serializeObject(prvfilePart2, prv.prv2);
}

/**
```

```
 * Enc.
 *
 * @param pubfile
 *          the pubfile
 * @param policy
 *          the policy
 * @param inputfile
 *          the inputfile
 * @param encfile
 *          the encfile
 * @throws Exception
 *           the exception
 */
public void enc(String pubfile, String policy, String inputfile, String encfile) throws
    Exception {

      AbePub pub;
      AbeCph cph;
      AbeCphKey keyCph;

      byte[] plt;
      byte[] cphBuf;
      byte[] aesBuf;
      byte[] pub_byte;
      Element m;

      /* get AbePub from pubfile */
      pub_byte = Common.suckFile(pubfile);
      pub = SerializeUtils.unserializeBswabePub(pub_byte);

      // ObjectInputStream inputStream = Utility.deSerializeObject(pubfile);
      // pub = (AbePub) inputStream.readObject();

      keyCph = Abe.enc(pub, policy);
      cph = keyCph.cph;
      m = pub.p.getGT().newElement();
```

```
        m = keyCph.key.duplicate();
        System.err.println("m = " + m.toString());

        if (cph == null) {
                System.out.println("Error happened in enc");
                System.exit(0);
        }

        cphBuf = SerializeUtils.bswabeCphSerialize(cph);

        // cphBuf = Utility.objectToByteArray(cph);

        /* read file to encrypted */
        plt = Common.suckFile(inputfile);
        // ObjectInputStream inputStream2 =
        // Utility.deSerializeObject(inputfile);
        // inputStream2.readFully(plt);
        aesBuf = AESCoder.encrypt(m.toBytes(), plt);

        Common.writeCpabeFile(encfile, cphBuf, aesBuf);
}

/**
 * dec_with_revocation_check.
 *
 * @param pubfile
 *          the pubfile
 * @param prvfilePart1
 *          the prvfile part1
 * @param encfile
 *          the encfile
 * @param m_decfile
 *          the m_decfile
 * @param user_id
 *          the user_id
 * @throws NoSuchDecryptionTokenFoundException
 * @throws AttributesNotSatisfiedException
 * @throws IOException
 * @throws ClassNotFoundException
 * @throws Exception
 *            the exception
 */
public void dec_with_revocation_check(String pubfile, String prvfilePart1, String encfile,
    String m_decfile,
              String user_id) throws AttributesNotSatisfiedException,
                  NoSuchDecryptionTokenFoundException, IOException {

      byte[] cphBuf;
      byte[] prv_1_byte;
      byte[] pub_byte, m_dec_byte;
      byte[][] tmp;

      AbeCph cph;
      AbePub pub;
      AbePrvPart1 prvPart1;
      AbeMDec mDec;

      /* get AbePub from pubfile */
      pub_byte = Common.suckFile(pubfile);
      pub = SerializeUtils.unserializeBswabePub(pub_byte);

      // ObjectInputStream inputStream = Utility.deSerializeObject(pubfile);
      // pub = (AbePub) inputStream.readObject();

      /* read ciphertext */
      tmp = Common.readCpabeFile(encfile);
      cphBuf = tmp[1];
      cph = SerializeUtils.bswabeCphUnserialize(pub, cphBuf);

      /* get AbePrvPart1 form prvfilePart1 */
      prv_1_byte = Common.suckFile(prvfilePart1);
      prvPart1 = SerializeUtils.unserializeBswabePrvPart1(pub, prv_1_byte);
```

```
        // ObjectInputStream inputStream2 =
        // Utility.deSerializeObject(prvfilePart1);
        // prvPart1 = (AbePrvPart1) inputStream2.readObject();

        mDec = Abe.m_dec(pub, prvPart1, cph);

        m_dec_byte = SerializeUtils.serializeBswabeMDec(mDec);
        Common.spitFile(m_decfile, m_dec_byte);

        // Utility.serializeObject(m_decfile, mDec);
}

/**
 * Dec.
 *
 * @param pubfile
 *            the pubfile
 * @param prvfilePart2
 *            the prvfile part2
 * @param encfile
 *            the encfile
 * @param m_decfile
 *            the m_decfile
 * @param decfile
 *            the decfile
 * @param user_id
 *            the user_id
 * @throws Exception
 *             the exception
 */
public void dec(String pubfile, String prvfilePart2, String encfile, String m_decfile, String
    decfile,
            String user_id) throws Exception {

        byte[] aesBuf, cphBuf;
```

```
byte[] plt;
byte[] prv_2_byte;
byte[] pub_byte, m_dec_byte;
byte[][] tmp;

AbeCph cph;
AbePub pub;
AbePrvPart2 prvPart2;
AbeMDec mDec;

/* get AbePub from pubfile */
pub_byte = Common.suckFile(pubfile);
pub = SerializeUtils.unserializeBswabePub(pub_byte);

// ObjectInputStream inputStream = Utility.deSerializeObject(pubfile);
// pub = (AbePub) inputStream.readObject();

/* read ciphertext */
tmp = Common.readCpabeFile(encfile);
aesBuf = tmp[0];
cphBuf = tmp[1];
cph = SerializeUtils.bswabeCphUnserialize(pub, cphBuf);

/* get AbePrvPart2 form prvfilePart2 */
prv_2_byte = Common.suckFile(prvfilePart2);
prvPart2 = SerializeUtils.unserializeBswabePrvPart2(pub, prv_2_byte);

// ObjectInputStream inputStream2 =
// Utility.deSerializeObject(prvfilePart2);
// prvPart2 = (AbePrvPart2) inputStream2.readObject();

m_dec_byte = Common.suckFile(m_decfile);
mDec = SerializeUtils.unserializeBswabeMDec(pub, m_dec_byte);

// ObjectInputStream inputStream3 =
// Utility.deSerializeObject(m_decfile);
```

```
            // mDec = (AbeMDec) inputStream3.readObject();

            if (mDec != null) {
                   Element m = pub.p.getGT().newElement();
                   m = Abe.dec(pub, prvPart2, cph, mDec).duplicate();

                   plt = AESCoder.decrypt(m.toBytes(), aesBuf);
                   Common.spitFile(decfile, plt);

            } else {
                   System.exit(0);
            }
      }
}
```