Privacy-Preserving Personal Health Record System Using Attribute-Based Encryption

by

Yao Zheng

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Electrical & Computer Engineering

by

June 2011

APPROVED:

Professor Wenjing Lou, Major Thesis Advisor

Professor Xinming Huang

Professor Shucheng Yu

Abstract

Personal health record (PHR) service is an emerging model for health information exchange. It allows patients to create, manage, control and share their health information with other users as well as healthcare providers. In reality, a PHR service is likely to be hosted by third-party cloud service providers in order to enhance its interoperability. However, there have been serious privacy concerns about outsourcing patients' PHR data to cloud servers, not only because cloud providers are generally not covered entities under HIPAA, but also due to an increasing number of cloud data breach incidents happened in recent years.

In this thesis, we propose a privacy-preserving PHR system using attribute-based encryption (ABE). In this system, patients can encrypt their PHRs and store them on semi-trusted cloud servers such that servers do not have access to sensitive PHR contexts. Meanwhile patients maintain full control over access to their PHR files, by assigning fine-grained, attribute-based access privileges to selected data users, while different users can have access to different parts of their PHR. Our system also provides extra features such as populating PHR from professional electronic health record (EHR) using ABE.

In order to evaluate our proposal, we create a Linux library that implement primitive of key-policy attribute-based encryption (KP-ABE) algorithms. We also build a PHR application based on Indivo PCHR system that allow doctors to encrypt and submit their prescription and diagnostic note to PHR servers using KP-ABE. We evaluate the performance efficiency of different ABE schemes as well as the data query time of Indivo PCHR system when PHR data are encrypted under ABE scheme.

Acknowledgements

It is with immense gratitude that I acknowledge all the people who have helped and inspired me during my Master studies.

I wish to thank, first and foremost, my advisor, Professor Wenjing Lou, for her guidance during my research and studies at WPI. With her enthusiasm and her rigorous attitude of scholarship, she showed me how to concentrate during my research. This thesis would not have been possible without her sound advice and encouragement.

My sincere thanks also goes to rest of my thesis committee: Professor Xinming Huang and Professor Shucheng Yu, for taking time out of their extremely busy schedule, and for their insightful comments, and hard questions.

I owe my deepest gratitude to my ESL teacher, Mr Billy McGowan, for his kindness and patient teaching. Throughout my thesis-writing period, he offers tremendous help with my writing technique and presentation rehearsal.

I am indebted to many of my fellow lab mates, Ming Li, Zhenyu Yang, Ning Cao, Qiben Yan, for making Cyber Security Laboratory a convivial place to work. In particular, I would like to thank Dr. Ming Li for his friendship and help in the past nine months. I hope he has a wonderful and successful time at Utah State University.

Last and most importantly, I wish to thank my parents, Weihua Zheng and Fengxian Guan, for giving birth to me, raising me, supporting me, teaching me, and loving me. I also would like to express my gratitude to my lovely fiancée, Qiumin (Celia) Peng, for her angelic smile, her tolerance, her inspiration and her love through all these year.

I remember it was a harsh winter when I first came to WPI. It is because of these generous and kind people I can graduate in this lovely summer. To them, I dedicate this thesis.

Thanks to all the people I have met throughout my life. It is a great honor to become a member of Class 2011 WPI in the year I am 26.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The term personal health record (PHR) has undergone substantial changes along with the emergence of cloud computing. In a relatively broad description, put forward by the Markle Foundation, A PHR is a set of computer-based tools that allow people to access and coordinate their lifelong health information and make appropriate parts of it available to those who need it [KJJ$^+$08]. We recognize that the Markle Foundation description has successfully predicted the evolving of PHR in the past ten years. Most healthcare information technology vendors and healthcare providers started their PHR services as a simple storage service, and then turn them into a complicated social-network like service for patients to share personal health information with others.

Currently, interest and investment in PHRs are usually motivated by goals of efficiency, increasing patient empowerment, or improving disease management. However, patients' greatest concern about PHRs, as well as other healthcare system, is security and privacy. The Health Insurance Portability and Accountability Act (HIPAA) of 1996 outlined the legal protections for PHR privacy and security. But, it does not address all the issues involved, especially because HIPAA only applies to covered entities including health plans, healthcare clearinghouses, and healthcare providers. Emerging cloud-based PHR service provider like Dossia, Microsoft, and Google are not covered entities.

Therefore, by introducing cloud computing into PHR service, several important issues regarding PHR privacy and security need better evaluation. Potentially, PHR could protect patient privacy and security in ways that are much more secure than traditional paper-based patient records, since it can provide additional security feature such as password protecting and audit tracking. However, by outsourcing PHR into a cloud server, patients lose physical control to their own healthcare data. PHRs residing on a cloud server are subject to more malicious insider and outsider attacks than paper-based records, which exist in only a small number of physically accessible locations. Hence, we argue extra steps must be taken to provide strong privacy assurance other than directly placing those sensitive data under the control of cloud servers. One straightforward solution is encrypting sensitive data before outsourcing it into cloud server. However, applying traditional cryptography scheme on a PHR system present a major barrier to access and share PHR. PHR system users need to deal with complicated key management problem to achieve fine-grained access control when their PHRs are encrypted using symmetric key cryptography or asymmetric key cryptography [LYRL10].

1.1 Research Goals

In this thesis, the fundamental goal is to propose and implement a practical design to achieve fine-grained data access control of PHR data in a semi-trusted cloud computing environments. We demonstrate PHR privacy issue can be partially solved by reducing it to the underlying cryptographic and key management problem. Relying on the novel one-to-many cryptography scheme, such as attribute-based encryption (ABE), we wish to construct a PHR architecture that aims to meet the following desiderata:

End-to-end Encryption. In a cloud computing paradigm, we tend to assume the physical servers of cloud-based systems to be semi-trusted comparing to centralized servers behind the firewall, in that they are subjected to more malicious inside, or outside attacks, than the later one. As a result, our approach is designed to secure PHR records from the point of origin (PHR data owner) all the way to the recipient (PHR data user) in an encrypted format.

Patient-Centric. In our system, patients should have full control of their medical records and can effectively share their health data with a wide range of users. In a cryptography sense, that means patients shall generate their own decryption keys and distribute them to their authorized users.

Collusion-Resistant. In our setting, PHR data can be accessed by multiple users, such as healthcare provider, health insurer, family member etc. Hence, we cannot neglect the possibility that these users may intentionally or unintentionally collude together to gain access to part of PHR data they do not have right to access separately. For that reason, in our design, the PHR data should remain confidential under such a circumstance.

Revocation and Delegation. A PHR system is highly dynamic. Much like a social network, patients can terminate their relation with certain PHR data user, such as a health insurer, indefinitely. In other word, patients should always retain the right to revoke access privileges and its corresponding decryption key when they fell necessary. Nevertheless, data users may have the need to grant temporally part of their access right to other parties. For example, a health insurer might only allow its accounting department to access part of customers' PHR data. As a result, we should also provide a delegation mechanism in our construction.

In this research, we will focus on the design and implement of a PHR system using proper cryptographic scheme. To validate our architecture, we also evaluate the applicability and efficiency of our construction.

1.2 Related Works

Several PHR systems have been proposed or implemented to enable access control on PHR. We classify them into two categories according to their different access control mechanisms.

Authentication-Based PHR system

Some PHR systems choose an attribute-based access control (ABAC) scheme or a role-based access control (RBAC) scheme to manage users' access right. This type of system usually places full trust on the cloud server where the PHRs reside in. A typical example of authentication-based PHR system is Indivo X platform [ASZ$^+$10]. Indivo is an open-source open standard personally controlled health record (PCHR) system that enables patients to own and manage their health records. Indivo provides patients the ability to share their records with different physicians, hospitals and clinics while maintaining access control properties on the patients' health records. Access control decisions are made by the Indivo server according to institutional policies and patient specified policies.

Cryptography-Based PHR system

Other PHR systems use cryptographically enforced access control scheme. This type of system usually allows patients to encrypt their PHR data and distribute corresponding decryption key to authorized user. A typical example of cryptography-based system is iHealthEMR [ALG$^+$]. It implements a self-protecting electronic medical records (EMRs) using attribute-based encryption. In that system, patient can encrypt each node in the XML-based EMR file with an automatic generated access policy before exporting it to cloud system. PHR users' access rights are defined by the attributes within their private key. However, it does not solve practical problems such as key revocation and key delegation. Nevertheless, the actual implementation is limited since the encrypted XML file contains malformed metadata and, therefore, cannot be accepted by the third party cloud service such as Google Health.

In our research, we demonstrate that authentication-based system can adopt our crypto-based PHR architecture by adding proper cryptographic operation into the authentication process. Therefore, our techniques can be designed to augment a authentication-based system, providing finer-grained protections and access control without the requirement of a honest cloud server. For cryptography-based PHR system, we demonstrate that attribute-based encryption (ABE) is more suitable to achieve patient-centric and fine-grained access control. We also extend other people's work by implementing a prototype security PHR application based on ABE. To facilitate our implementation, we have built a Linux library to support key-policy attribute-based encryption (KP-ABE) scheme.

1.3 Research Outline

We first give a comprehensive overview of PHR system. Next, we examine one particular PHR system, Indivo X platform, which our implementation is based upon. We focus on the Indivo API which is meant to help health care providers develop modules and specialized applications that leverage the platform services.

In chapter 3, we discuss in detail of one-to-many cryptography schemes such as ABE. We shall see that ABE is quite practical and compare well to other cryptography scheme when addressing PHR related issue.

In chapter 4, we will present our PHR system model, our KP-ABE library and our Indivo PHR application. In order to validate the practicality of applying ABE PHR system, we evaluate the timing consumption of ABE operation and data query time of our Indivo PHR application.

Finally, in chapter 5, we will discuss the open problems and future work.

This thesis is intended to be self-contained. It is hoped that this work will be a useful guide to implementing cryptography based PHR system to researchers who have little experience with cryptosystems. Therefore, we also take sometime to explain bilinear maps and secret sharing in chapter 2 to so reader can understand better about attribute-based encryption (ABE) without referring to any other sources.

We are currently in the process of publishing our implementation source code. More explanation about the algorithms and feature can be obtained by directly emailing the author at zhengyao@wpi.edu

Chapter 2

Personal Health Record Systems

Technologies such as PHR play an essential role during digital transformation of healthcare [KPB06]. In stimulus bill, president Obama envisions "the utilization of a certified electronic personal health record for each person in the United States by 2014." However, to realize the potential value of PHR and PHR, significant improvements are needed in the areas of privacy, security, and interoperability.

In this Chapter, we would like to outline the value of PHR and PHR system and give formal definitions of PHR and PHR system. We further characterize PHR and PHR system according to their attributes.

We provide a short discussion of PHR system and cloud computing. We will see a PHR system becomes more valuable when it connects to more healthcare information systems. That makes a PHR system more suitable for cloud-based deployment.

We study the privacy and security aspects of PHR system. Unique privacy issues arise in relation to PHR systems offered by third parties, including some emerging cloud-based service that warehouse and mine personal health data for secondary uses.

The Indivo project is perhaps the most famous inversion of the current approach to PHR, in that the record resides with the patients and the patients grant permissions to institutions, clinicians, researchers, and other users of medical information. Nevertheless, the Indivo X API, built to public standards, enable us to quickly create and integrate our cryptographic implementation into patients' online records.

2.1 PHR and PHR System Description

The definition of PHR is heterogeneous and evolving. One of the challenges in delineating PHR research is finding a consistent description of what PHR actually entails. Markle Foundation defines PHR as a set of computer-based tools that allow people to access and coordinate their lifelong health information and make appropriate parts of it available to those who need it. In some concepts, the PHR includes the patients' interface to a healthcare providers' electronic health record (EHR). In others, PHR are any consumer/patient-managed health record.

Note that people tend to define new concepts in the context of existing technologies. we can give another definitions of PHRs and PHR system by referring to the terms "electronic health record (EHR)" and "electronic health record systems (EHR system)" which have been adopted by the standards development organization HL7. The term "PHR" refers to the collection of information about and individual's health and health care, stored in electronic format. The term "PHR system" refers to the addition of computerized tools that help an individual understand and manage the information contained in a PHR.

A better description of PHR and PHR systems can be established by characterizing them according to their attributes. A 2005 report [1] from National Committee

[1]National Committee on Vital and Health Statistics, *Personal Health Records and Personal Health Record Systems* (Washington, DC: U.S. Department of Health and Human Services, Febru-

on Vital and Health Statistic (NCVHS) outlines the attributes about PHR and PHR systems:

Scope and Nature of Content. All PHR systems have consumer health information, personal health journals, information about benefits and/or providers. Some PHR system have clinical information (such as lab reports).

Source of Information. PHR data may come from the patient, caregiver, healthcare provider, payer, etc. Some PHRs are populated with data from EHRs.

Features and Functions. PHR systems offer a wide variety of features, including the ability to view personal health data, exchange secure messages with providers, schedule appointments, renew prescriptions and enter personal health data; decision support (such as medication interaction alerts or reminders about needed preventive services); the ability to transfer data to or from an electronic health record; and the ability to track and manage health plan benefits and services.

Custodian of the Record. The PHR record may be operated by a number of parties, including the consumer or patient, and independent third party, a healthcare provider, an insurance company or an employer.

Data Storage. Data may be stored in a variety of locations, including an Internet-accessible database, a provider's EHR, the consumer/patient's home computer, a portable device such as a smart card or thumb drive, or a privately maintained database.

Party Controlling Access to the Data. While consumers or patients always have access to their own data, they do not always determine who else may access

ary 2006), p. 15; available at `www.ncvhs.hhs.gov/0602nhiirpt.pdf`.

it. For example, PHRs that are “views” into a provider’s EHR follow the access rules set up by the provider. In some cases, consumers do have exclusive control.

From these attributes, we may compare a PHR system to a hub and spoke model. It aggregates data from multiple source (insurance companies, hospitals, PBMs, labs, and the patient) into centralized, patient-controlled data repositories. It is also responsible for making the record available to authorized users.

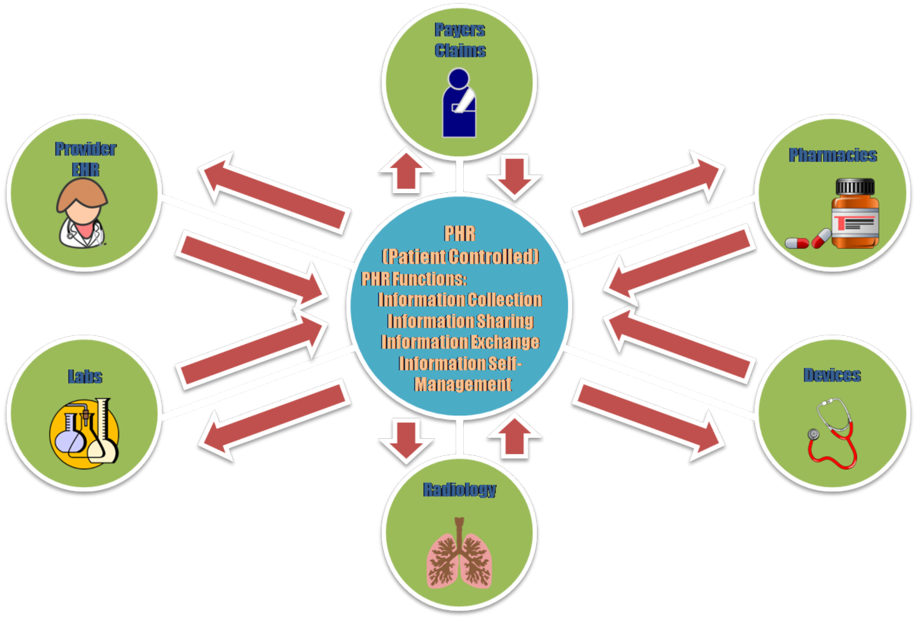


Figure 2.1: Hub and Spoke Model of PHR system

2.2 PHR System and Cloud Computing

There are four emerging PHR system. Based on the primary source of data for the PHR, they are defined as provider-tethered, payer-tethered, third-party/free standing, and interoperable PHR system. All of them can be derived from the hub and spoke model above [KJJ$^+$08]. For example, a provider-tethered or a paytethered PHR system can be considered in the hub and spoke model with just one thick spoke (provider-tethered PHR system are tied to a healthcare organizations internal record system; payer-tethered systems are tied to a given payers system). A third-party/free standing PHR system can be considered in the hub and spoke model without any spoke (consumers act as relays in third-party/free standing PHR system to aggregate data from different, unconnected sources). An interoperable PHR system can be considered as a full version of hub and spoke model.

According to the hub and spoke model, interoperability represents a key component of PHR system. If a PHR system cannot exchange data with other healthcare systems, PHR will become isolated from other healthcare information, with limited access and transient value [KJJ$^+$08]. Therefore, the minimal requirements of a PHR system are being capable of exporting data to and importing data from other systems in a standardized way. More advanced PHR system in the future will function as seamlessly integrated, interoperable subsystem of other health systems [LSW10].

In order to build a PHR system with high interoperability, we need to assess the limitation of the standard we intend to use in a PHR system. The healthcare industry has developed several standards through which healthcare data can be transferred among different information systems. These standards include and are not limited to health language seven (HL7), health insurance portability and accountability (HIPAA), electronic data interchange (EDI) X12 Version 4010, continuity of care record (CCR) and continuity of care document (CCD), etc. Current PHR systems usually support multiple healthcare information standards, which make it possible for PHR systems to interoperate with other systems by providing standardized interface between different healthcare systems.

A PHR system can be built upon various types of infrastructure, such as, personal computer, portable device, Internet, etc. Among them, cloud based deployment offers significant advantages over others underlying infrastructure .Deploying a PHR system under cloud environment maximize the possibility for PHR system to interoperate with other systems throughout the entire health information environment. From a technical perspective, cloud-based PHR systems offer new possibility, such as ubiquitously accessible to the nomadic user, elastic computation resource for data mining service, easily development and deployment of new applications, high-degree of fault tolerance, etc., all without the concern of capacity and location of the actual infrastructure [Kau09].

GoogleHealth, Microsoft HealthVault and Dossia solutions are the first steps in the direction of building PHR systems in a cloud environment. However, beside the security and privacy concern we will address later, the other major issue that prevents putting their solution into commercial use is the inability to upload patient health record directly from healthcare provider. Right now, patients need to obtain their electronic medical records (EMR), which is the main source of information that feed the PHR, from their healthcare provider before manually importing them into the PHR system. Often, it takes several weeks before patients can access their medical records from their healthcare providers, therefore, limiting the usage of PHR system. However, such PHR adoption related issue can be solved based on the nature of cloud computing. In fact, GoogleHealth does offer interface with localized EMR database, even though the adoption rate among healthcare providers is fairly low due to the concern about extra training.

2.3 Privacy and Security of PHR System

Like nearly every type of electronic healthcare applications, patients' greatest concern about PHR system is security and privacy. A new national consumer survey for the California HealthCare Foundation shows that seventy-five percent of people choose not to use PHR system because they are very concerned about the privacy and security of their personal health information. Sadly, embracing cloud-based PHR system architecture seems to aggravate such problem. This is not surprising, given a number of recent highly publicized personally identifiable electronic information thefts target cloud-based service. When designing a cloud-based PHR system, the following issues regarding privacy and security need better evaluation.

Who Is In Control

By adopting cloud-based PHR systems, patients face a dilemma. On one hand, they entrust their healthcare data to cloud service providers such as Google and Microsoft. On the other hand, according to HIPPA, cloud service providers are not "covered entities", therefore, not obligated to ensure the confidentiality and proper access to consumers' PHR. As a result, there is a natural conflict between cloud service providers' legal responsibility and their legal obligations when they maintain patients' PHRs. For that reason, we define the underlying infrastructure (physical server in particular) of cloud-based PHR system to be semi-trusted. Any protocol it follows should be secure in an honest but curious (HBC) sense. So that, were any malicious outside or inside attack happen, PHR systems shall not reveal any privacy information within the HBC setting. And PHR systems shall not be held responsible for any privacy information leakage beyond the HBC setting. In this setting, patients claim full control over their PHR by only revealing non sensitive information to PHR systems.

Traditional authentication-based PHR systems protect patient's privacy by encrypting PHR. However, it is usually the systems themself that handling the decryption keys. Not only it is not secure in an HBC setting, it is also difficult to achieve fine-grained access control. Password management systems, such as Last-Pass, address this problem by not processing a user's master decryption key. While their solution is intriguing, PHR databases are far more complex than a password database, therefore, require more advance cryptography technology.

How To Be In Control

Referring to the definition of PHR, it seem clear that patients should have exclusive control over sharing and accessing the information in their PHR. However, the actual situation might be complicated. Usually, PHR data come from various source, including healthcare provider, payer, caregiver, etc. Even though these data become part of patients' personal properties the minute they are merge into patient's PHR, the sharing and accessing policy made by the original data provider can still be valuable in order to maintain the interoperability of PHR system. Consider the following case. A hospital exports patients' emergency medical informations to the PHR system. The original sharing policy made by the hospital is allowing all emergency medical technician (EMT) to access it during emergency situation. However, if patients do not have the original policy as reference, they might create a policy not allowing any EMT to access these emergency medical informations. This new policy practically diminishes the value of emergency information. Ideally, a PHR system should inherit the sharing policy made by the data source while allowing patients to modify it according to their need.

Another issue that worth mentioning is the veracity of the patients' PHR information. Data aggregated within the PHR system are the property of patients. They are certainly allowed to annotate, hide any document in the record. However, for those documents that are not originally created by the patient, such as doctor prescription, actions like these might be considered as healthcare fraud. In this case, personal control should not extended as far as document content. In other words, PHR system must provide healthcare providers with an accurate and trustworthy view into the patient's health data.

Clearly, critical tradeoffs exist between privacy, security and usability. The tighter you grip, the more accessibility you lose. That is also the main reason why most authentication-based PHR systems are reluctant to adopt stronger privacy and security measures.

2.4 Case study: Indivo

Among all existing PHR systems, The Indivo project, developed by Children's Hospital, in Boston, is the most successful example. Micky Tripathi, President and CEO of the Massachusetts eHealth Collaborative, noted that "Indivo is the most thoughtful and firmly grounded PHR project in the country, bar none." Currently, Indivo is actively deployed as the backbone of various commercial PHR system, in particular Boston Children's Hospital and the Dossia Consortium. Indivo is also the underline platform of our implementation. For that reason, we would like to briefly describe the key features of Indivo.

The original Indivo project, namely personal internetworked notary and guardian (PING), was initiated in 2005 [SMK05]. It means to build an open-source, public standard platform that enables patients to assemble, maintain and manage their personally controlled health records [KPB06] (PCHRs, a PCHR usually refer to the subset of personal health records which are populated from professional data such as EMR). Right from Indivo 1.0 release, it was designed as a distributed, cloud-based, online system which provides an open application programming interface (API) for connecting to user specific personal health applications (PHAs). The key components in Indivo include the Indivo API, encrypted storage and staging server [MSCA07]. In the latest release, Indivo X, the project team significantly enriches and stabilizes the Indivo API. As a result, like most useful Web APIs, such as Amazon S3, Facebook, Indivo now become a platform with feature-level substitutability which enables developer to quickly design and integrate novel feature into the base system [ASZ+10].

Indivo API

The Indivo X application programming interface (API), built to support feature-level substitutability, is the contract between the PCHR platform and the end-user applications [ASZ+10]. The need for deep customization is inherent to the PCHR concept. Patients have different needs for their PCHR experiences. For example, a patient with diabetes usually prefers application tailored to highlight glucose measurement rather than generic laboratory application. Therefore, instead of an expansive and complex interface to suit each patient's need, Indivo X API provides a minimal, focused feature set based on real-world requirement. These feature are based on the experience gain from each real-world deployment, including a patient survey engine, a genomic messaging system, and a patient-social-network integration [ASZ+10] (this feature is crucial to our implementation since it provides each application with its own storage via the Indivo API). With this extensive customizability, patients can easily customize the feature to meet their need.

In Indivo X 1.1, the API converged on approximately 80 simple calls including administration, data reports, sharing, messaging, data entry and versioning [ASZ+10]. Comparing to previous Indivo API, a lot of useful features are now included in the API. For example, during our implementation, we used to depend on a particular PHA, called "subscription agent" to identify patient's EMR and convert it into CCR

XML file. Now the functions of "subscription agent" are included in the Indivo X API.

The Indivo API abides by the REST design pattern. Any PHAs, which wish to communicate with Indivo platform, make HTTP calls to the Indivo API endpoint using the REST convention. This call is then authenticated using Oauth protocol.

Data storage and Data model

Inside Indivo, the PCHRs are stored on a separate backend server in a encrypted format. The server provides XML REST services for authorized PHAs. Unlike Microsoft Health Vault and Google Health, all data within Indivo are stored in XML documents, including core Indivo schemas, supported standard, etc. Further, each PCHR in the system is fractured into loosely-related data packets to mask the size of patients' record [SMK05]. When there is a request, for example, "Allergy Report", from the PHAs, the server will pull the basic information, such as data of onset out from related documents. This information then will be filled into a template XML file to provide one aggregated XML report. Therefore, in this report, each individual allergy includes only basic fields, with a reference to individual documents which contain more detail.

Indivo uses PostgreSQL relation database for robust data storage. In order to maximize system performance, it uses database level encryption to achieve data security. This does not protect against someone who has access to the data (such as the server itself). Even though it can achieve tighter security by applying column level encryption or table level encryption, it is still not the desired way to enforce patient-centric, fine-grain access control. First of all, the decrypted data and the decryption keys are present on the server for a brief time while they are being decrypted and communicated between the clients and servers. This presents a brief moment where the data and keys can be intercepted by semi-trusted servers. Therefore, it is not secure in HBC setting. Nevertheless, column level encryption or table level encryption usually use symmetric keys encryption (such as AES_256) to directly encrypt the data. Then it use certificates and/or asymmetric keys to protect the symmetric keys. However, current SQL server can only generate asymmetric key through traditional public cryptography schemes (such as RSA_1024). In this setting, If authenticated by patients, A PHA will have privilege to access all of their PHRs. Nevertheless, patients have to management multiple asymmetric key pairs in order to achieve fine-grain access control.

Beside the Indivo API and Indivo backend storage server, the Indivo team also developed an Indivo staging server. The staging server essentially provides the ront-end UI and web frame [ASZ+10]. Using the Indivo X API and the staging server, developers can quickly prototype a new research quality PHA, focused on a small feature set. It is equally possible to create a PHA without the staging server, though the work required is more challenging.

Chapter 3

Attribute-Based Encryption

Our proposed PHR system is build upon the attribute-based encryption (ABE) scheme. The concept of ABE was introduced along with another cryptography called fuzzy identity-based encryption (FIBE) [SW05] by Sahai and Waters. Both schemes are based on bilinear maps (pairing). Since the main goal in FIBE is error tolerance, it only supports access structure in the shape of a threshold gate. ABE, on the other hand, support every linear secert sharing (LSSS) realizable access structure.

Therefore, we can best understand ABE by first introducing bilinear maps and LSSS. Knowledge of finite fields and elliptic curves is required to better explain the actual implementation of cyclic groups in bilinear maps. However, a extensive investigation of these two area is beyond the scope of this thesis. Nevertheless, thank to the contribution of Ben Lynn. [Lyn07], we can implement a attributes-based encryption scheme without learning much about elliptic curves or number theory.

We discuss two ABE scheme, namely key-policy attribute-based encryption (KP-ABE) [GPSW06] and ciphertext-policy attribute-based encryption (CP-ABE) [BSW07].

The essential difference between this two schemes is whether the system use attributes to describe the encrypted data or the user's private key. Generally speaking, CP-ABE is conceptually closer to RBAC, while KP-ABE is more closer to ABAC.

3.1 Bilinear Maps

There are several definitions of bilinear maps, or pairings, in a cryptography paradigm. The differences between the expressions of these definitions are subtle. Yet, the differences between the underline cryptography meaning of these definitions are significant. Generally speaking, bilinear maps associates pairs of elements from two algebra group to to yield an element of a third algebra group that is linear in each of its arguments. According to Ben Lynn [Lyn07], the essential property of bilinear maps is that they give cyclic groups additional properties. We outline three definitions of bilinear maps in order of restrictiveness.

Symmetric Pairing

Definition 3.1.1. Let G, G_T be cyclic group of prime order p. Let g be a generator of G. A *symmetric bilinear pairing* or *symmetric bilinear map* e is an efficiently computable function

$$e : G \times G \to G_T$$

such that

(i) (Nondegeneracy) $e(g, g) \neq 1$

(ii) (Bilinearity) $e(g, g)^{ab} = e(g, g)^{ab}$ for all $a, b \in \mathbb{Z}$

The symmetric pairing is the original and simplest definition of bilinear maps. It shows some important properties of bilinear maps. For example, the decisional

Diffie-Hellman problem is easy to solve in bilinear maps, since $z = xy$ if and only if $e(g, g^z) = e(g^x, g^y)$. However, beside certain suitable supersingular elliptic curves, symmetric pairing can not be instantiated from most of ordinary elliptic curves.

Asymmetric Pairing

Definition 3.1.2. Let G_1, G_2, G_T be cyclic group of prime order p. Assume the CDH is hard in G_1. Let $\phi : G_2 \to G_1$ be an efficiently computable group isomorphism Let g_2 be a generator of G_2. Set $g_1 = \phi(g_2)$ (so g_1 generates G_1). A *bilinear pairing* or *bilinear map* e is an efficiently computable function

$$e : G_1 \times G_2 \to G_T$$

such that

(i) (Nondegeneracy) $e(g_1, g_2) \neq 1$

(ii) (Bilinearity) $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ for all $a, b \in \mathbb{Z}$

The asymmetric pairing looses the definition of symmetric pairing in order to use a wider range of ordinary elliptic curves. However, there is a problem with hashing in this definition. It is no known how to hash a string to an element of G_2 such that its discrete log to some fixed base is known. As a result, this definition can complicate the design for some cryptography systems.

The General Bilinear Pairing

Definition 3.1.3. Let G_1, G_T be cyclic group of prime order p. Let G_2 be a group where each element has order dividing r. In particular G_2 is not necessarily cyclic. A *bilinear pairing* or *bilinear map* e is an efficiently computable function

$$e : G_1 \times G_2 \to G_T$$

such that

(i) (Nondegeneracy) $e(g_1, g_2) \neq 1_{G_T}$ for all $g_2 \in G_2$ if and only if $g_1 = 1_{G_1}$, and $e(g_1, g_2) \neq 1_{G_T}$ for all $g_1 \in G_1$ if and only if $g_2 = 1_{G_2}$.

(ii) (Bilinearity) for all $g_1 \in G_1$ and $g_2 \in G_2$ and $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ for all $a, b \in \mathbb{Z}$

The general bilinear pairing is the most flexibility. It solve the hash problem by reduce G_2 to be not necessarily cyclic. However, in this setting, the hardness assumption must be changed. Based on different scheme, we have to assume certain problems are hard in both groups. For example, we can assume that given $g_1, g_1^x \in G_1$ and $g_2 \in G_2$, there is no efficient algorithm to compute g_2^x.

Cryptography Implementation

A cryptography system usually choose the most general definition first, which allow maximize choice for the pairing during implementation. Then one may have to pick the extra assumption if a more restrictive definition is required. Note that the more restrictive the definition is, the less pairing families one can choose. In practical terms, G, or G_1 and G_2 are always groups of points on an elliptic curve, and G_T is always a subgroup of multiplicative group of a finite field.

3.2 Linear Secret-Sharing Scheme

The idea of linear secret-sharing scheme (LSSS) and monotone span programs was discussed by Amos Beimel [Bei96]. In a LSSS, dealer holds a secret and distributes the shares of the secret to parties. Parties can reconstruct the secret from a linear combination of the shares of any authorized set. A famous example of LSSS is the Shamir t-out-of-n threshold scheme. In that scheme, the hardness of secret reconstruction depends on the hardness of polynomial reconstruction. We briefly reiterate some of the definitions from Beimel's work to formally define a LSSS.

Definition 3.2.1. [Access Structure]: Let $\{P_1, ..., P_n\}$ be the set of parties. A collection $\mathcal{A} \subseteq 2^{\{P_1,...,P_n\}}$ is monotone if $\forall B, C$: if $B \in C$ and $B \subseteq C$ implies $C \in \mathcal{A}$. An *access structure* is a monotone collection [1] $\mathcal{A}$ of non-empty subsets of $\{P_1, ..., P_n\}$ (that is, $\mathcal{A} \subseteq 2^{\{P_1,...,P_n\}} \backslash \{\emptyset\}$). The set in $\mathcal{A}$ are called the authorized sets, and the sets not in $\mathcal{A}$ are called the unauthorized sets.

Definition 3.2.2. [Secret Sharing]: Let S be a finite domain of secrets. A *secret-sharing scheme* realizing an access structure $\mathcal{A}$ is a scheme in which the input of the dealer is a secret $s \in S$ such that the following two requirements hold:

Reconstruction requirement The secret s can be reconstructed by any authorized set. That is, for any set [2] $G \in \mathcal{A}(G = \{i_1, ..., i_{|G|}\})$, there exists a reconstruction function $h_G : S_{i_1} \times ... \times S_{i_{|G|}} \to S$ such that for every secret s, and every random input r,

$$\text{If } \Pi(s, r) = \langle s_1, s_2, ..., s_n \rangle \text{ then } h_G(s_{i_1}, ..., s_{i_{|G|}}) = s$$

Security requirement Every unauthorized set of parties cannot reveal any partial information about the secret. Which means: for any set $B \notin \mathcal{A}$, for every two secrets $a_1, a_2 \in S$, and for every vector of possible pieces $\{s_i\}_{i \in B}$:

$$\Pr[\bigwedge_{P_i \in B} \Pi_i(a_1, r) = s_i] = \Pr[\bigwedge_{P_i \in B} \Pi_i(a_2, r) = s_i]$$

Definition 3.2.3. [Linear Secret Sharing Schemes(LSSS)]: Let $\mathcal{K}$ be a finite field, and Π be a secret sharing scheme with domain of secrets $S \in \mathcal{K}$ realizing an access structure $\mathcal{A}$. We say that Π is a *linear secret sharing scheme* over $\mathcal{K}$ if:

[1]The monotone property of access structure is easy to understand. Because if $\mathcal{A}$ is a access structure, the superset of $\mathcal{A}$ should also be a access structure.

[2]if the secret sharing scheme require $\{G : |G| \geq t\}$ (n is the number of parties in the access structure), it is called a *t-out-of-n threshold secret sharing scheme.*

(i) The piece of each party is a vector over $\mathcal{K}$. That is, for every i there exists a constant d_i such that the piece of P_i is taken from $\mathcal{K}^{d_i}$. We denote by $\Pi_{i,j}(s,r)$ the *j-th* coordinate in the piece of P_i (where $s \in S$ is a secret and $r \in R$ is the dealer's random input).

(ii) For every authorized set, the reconstruction function of the secret from the pieces is linear. That is for every $G \in \mathcal{A}$ there exist constants $\{\alpha_{i,j} : P_i \in G, 1 \leq j \leq d_i\}$, such that for every secret $s \in S$ and every choice of random inputs $r \in R$,

$$s = \sum_{P_i \in G} \sum_{1 \leq j \leq d_i} \alpha_{i,j} \cdot \Pi_{i,j}(s,r)$$

where the constants and the arithmetic are over the field $\mathcal{K}$. The total size of the pieces in the scheme is defined as $d \triangleq \sum_{i=1}^{n} d_i$.

Definition 3.2.4. A secret sharing scheme is linear (linear generation of pieces) if:

(i) The piece of each party is a vector over $\mathcal{K}$.

(ii) During the generation of the pieces, the dealer chooses independent random variables, denoted $r_1,...r_l$, each one distributed uniformly over $\mathcal{K}$. Each coordinate of the piece of every party is a linear combination of $r_1,...r_l$, and the secret s.

Example 3.2.1. [Shamir t-out-of-n threshold scheme]: Let p^m be the size of the domain of secrets, where p^m is a prime-power bigger than n (the number of parties in the access structure). Let $s \in \mathbb{F}_{p^m}$ be the secret. The dealer chooses independently with uniform distribution $t-1$ random element in $\mathbb{F}_{p^m}$, which are denoted by $r_1,...r_{t-1}$. These elements and the secret s define a *secret polynomial*

$$p(x) \triangleq r_{t-1}x^{t-1} + r_{t-2}x^{t-2} + ... + r_1 x + s$$

Observe that $p(0) = s$. The dealer gives the piece $p(i)$ to party P_i ($i \in \mathbb{F}_{p^m}$). This piece is a linear combination of the random inputs and the secret. Now each set of cardinality at least t can reconstruct $p(x)$ by interpolation. That is, the set $\{P_{i_1}, ..., P_{i_t}\}$ holding the pieces $\{s_{i_1}, ..., s_{i_t}\}$, can compute the *Lagrange interpolation formula*

$$p(x) = \sum_{j=1}^{t} s_{i_j} \prod_{i_j \in S i_d \neq i_j} \frac{i_d - x}{i_d - i_j}$$

Where set $S = \{i_1, ..., i_t\}$. One can reconstruct the secret by substituting x with 0 in the formula. We define *Lagrange coefficient* $\Delta_{i_d,S}$ for i_d and set S.

$$\Delta_{i_d,S}(x) = \prod_{i_j \in S i_d \neq i_j} \frac{i_d - x}{i_d - i_j}$$

Shamir Threshold Scheme is linear because the secret is a linear combination of the pieces $\{s_{i_1}, ..., s_{i_t}\}$.

3.3 Key-Policy Attribute-Based Encryption

The key-policy attribute-based encryption (KP-ABE) was first introduced in 2006 by Goyal et al. In this cryptography system, ciphertexts are labelled with sets of attributes. Private keys, on the other hand, are associated with access structures $\mathcal{A}$. A private key can only decrypt a ciphertext whose attributes set is a authorized set of the private key's access structure, that is $\mathcal{A}(\gamma) = 1$. KP-ABE is a cryptography system built upon bilinear map and LSSS. Using the knowledge of previous sections, We can formally describe the construction of this cryptography system.

Define the Access Structure $\mathcal{A}$

In this case, the access structure $\mathcal{A}$ represent a tree. Each non-leaf node of the tree is a k_x-out-of-n_x threshold gate (a Shamir Threshold Scheme with $\mathbb{F}_{p^m} = \mathbb{Z}_p$). n_x is the number of children of node x. k_x is the gate's threshold value. The following four functions are defined to better explain node relation during construction.

(i) *parent(x)* return the parent of node x.

(ii) *att(x)* return attribute associated with node x when it is a leaf node.

(iii) *index(x)* return the index number of node x among all children of x's parent node.

(iv) *child(x,i)* [3] return the child of node x with index number equal to i .

Every non-leaf node x is related to its parent by inheriting one secret share of *parent(x)*.

Choose a Definition of Bilinear Map

KP-ABE uses symmetric bilinear maps ($G = \langle g \rangle$ with prime order p), while assuming BDDH to be hard in G.

Construction

KP-ABE Setup Define the attributes universe as $\mathcal{U} = \{1, 2, ..., n\}$. Associate each attribute $i \in \mathcal{U}$ with a number t_i chosen uniformly at random in $\mathbb{Z}_p^*$. Choose y uniformly at random in $\mathbb{Z}_p$. The public key is

$$PK = (T_1 = g^{t_1}, ..., T_1 = g^{t_{|\mathcal{U}|}}, Y = e(g, g)^y)$$

The master key is:

$$MK = (t_1, ..., t_{|\mathcal{U}|}, y)$$

[3]We add this function to the original definitions. This function is only used during our implementation.

KP-ABE Encryption (M, γ, PK) Choose a random value s in $\mathbb{Z}_p$. Encrypt a secret message M in G_T with a set of attributes γ. The ciphertext is:

$$E = (\gamma, E' = MY^s, \{E_i = T_i^s\}_{i \in \gamma})$$

KP-ABE Key Generation $(\mathcal{A}, MK)$ This algorithm output a private key D embedded with a access structure $\mathcal{A}$. The access structure $\mathcal{A}$ is realized by the following three steps:

(i) Each non-leaf node is defined as a Shamir Threshold Scheme. Set the degree of secret polynomial p_x to be $d_x = k_x - 1$.

(ii) For root node r, set $p_r(0) = y$. And randomly choose d_r element in $\mathbb{Z}_p$ to completely define p_r. For any other non-leaf node x, set its secret to be one secret share of its parent node, that is $p_x(0) = p_{parent(x)}(index(x))$. And randomly choose d_x element in $\mathbb{Z}_p$ to completely define p_x.

(iii) For each leaf node x, assign the following value

$$D_x = g^{\frac{p_x(0)}{t_{att(x)}}}$$

Let X be the set of leaf nodes in $\mathcal{A}$. The private key is

$$D = (\mathcal{A}, \forall x \in X : D_x = g^{\frac{p_x(0)}{t_{att(x)}}})$$

KP-ABE Decryption (E, D) The decryption algorithm is realized by the following three steps:

(i) For each non-leaf node x, all nodes z are children of x. Let S_x be an arbitrary k_x-sized set of child nodes z such that DecryptNode$(E, D, z) = F_z \neq \perp$, if S_x

does not exists, $\text{DecryptNode}(E, D, x) = F_x = \perp$

$$\begin{aligned} F_x &= \prod_{z \in S_x} F_z^{\Delta_{i,S'_x}(0)}, \text{ where } {}^{i=index(z)}_{S'_x=\{index(z):z\in S_x\}} \\ &= \prod_{z \in S_x} \left(e(g,g)^{s\cdot p_z(0)}\right)^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z \in S_x} \left(e(g,g)^{s\cdot p_{parent(z)}(index(z))}\right)^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z \in S_x} e(g,g)^{s\cdot p_x(i)\cdot\Delta_{i,S'_x}(0)} \\ &= e(g,g)^{s\cdot p_x(0)} \end{aligned}$$

(ii) For each leaf node x

$$\text{DecryptNode}(E, D, x) = \begin{cases} e(D_x, E_i) = e(g^{\frac{q)x(0)}{t_i}}, g^{s\cdot t_i}) = e(g,g)^{s\cdot p_x(0)} & \text{if } i = att(x) \in \gamma \\ \perp & \text{otherwise} \end{cases}$$

(iii) Apply this recursive algorithm to root node r, $\text{DecryptNode}(E, D, r) = e(g, g)^{ys} = Y^s$. Since $E' = MY^s$, the secret message is

$$M = \frac{E'}{Y^s}$$

In previous construction, the size of PK is linear to the size of $\mathcal{U}$. In the original paper, the authors present another construction which they call large attributes universe. In that construction, size of PK is only linear to the pre-defined maximum size of $\mathcal{U}$. The large universe construction is more practical since it does not require public key and master key update whenever a new attribute is added. We choose this construction because it is more straightforward to implement. Adopting large universe construction will be one of the future works in order to make the overall system more dynamic.

In KP-ABE scheme, delegation of private keys means converting the original access structure $\mathcal{A}$ into a more strict access structure $\mathcal{A}'$. In the original paper, the authors present a three-step delegation based on large universe construction. However, as we will see in the next section, delegating KP-ABE private keys is , by nature, more difficult compare to CP-ABE private key delegation.

3.4 Ciphertext-Policy Attribute-Based Encryption

The ciphertext-policy attribute-Based encryption (CP-ABE) was proposed approximately one year latter after KP-ABE. Not like KP-ABE, in CP-ABE, access structures $\mathcal{A}$ are built within ciphertexts, while private keys are associated with sets of attributes γ. We briefly describe CP-ABE construction as follow.

Define the Access Structure $\mathcal{A}$

The access structure $\mathcal{A}$, in CP-ABE, represents a tree. Each non-leaf node of the tree is a k_x-out-of-n_x threshold gate. n_x is the number of children of node x. k_x is the gate's threshold value.

Choose a Definition of Bilinear Map

CP-ABE uses symmetric bilinear maps ($G = \langle g \rangle$ with prime order p), while assuming BDDH to be hard in G.

Construction

CP-ABE Setup Choose two variable α, β uniformly at random in $\mathbb{Z}_p$. The public key is

$$PK = (g, h = g^{\beta}, f = g1/\beta, e(g, g)^{\alpha})$$

The master key is:

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

CP-ABE Encryption $(M, \mathcal{A}, PK)$ This algorithm output a ciphertext C embedded with a access structure $\mathcal{A}$. $\mathcal{A}$ is realized by the following three steps:

(i) Each non-leaf node is defined as a Shamir Threshold Scheme. Set the degree of secret polynomial p_x to be $d_x = k_x - 1$.

(ii) For root node r, choose a random value s in $\mathbb{Z}_p$. Set $p_r(0) = s$. And randomly choose d_r element in $\mathbb{Z}_p$ to completely define p_r. For any other non-leaf node x, set its secret to be one secret share of its parent node, that is $p_x(0) = p_{parent(x)}(index(x))$. And randomly choose d_x element in $\mathbb{Z}_p$ to completely define p_x.

(iii) define a hash function $H : \{0,1\}^* \rightarrow G$ which is modeled as a random oracle. For each leaf node x, assign the following value

$$E_x = g^{p_x(0)}, E'_x = H(att(x))^{p_x(0)}$$

Let X be the set of leaf nodes in $\mathcal{A}$. Encrypt a secret message M in G_T with the a access structure $\mathcal{A}$. The ciphertext is

$$E = (\mathcal{A}, \tilde{E} = Me(g, g)^{\alpha s}, \dot{E} = h^s, \forall x \in X : E_x = g^{p_x(0)}, E'_x = H(att(x))^{p_x(0)})$$

CP-ABE Key Generation (γ, MK) This algorithm will associate a set of attribute γ with a private key. Choose a random value r in $\mathbb{Z}_p$. Associate each attribute $i \in \gamma$ with a random number $r_i \in \mathbb{Z}_p$ The private key is:

$$D = (\dot{D} = g^{(\alpha+r)/\beta}, \forall i \in \gamma : D_i = g^r \cdot H(i)^{r_i}, D'_i = g^{r_i})$$

CP-ABE Decryption (E, D) The decryption algorithm is realized by the following three steps:

(i) For each non-leaf node x. All nodes z are children of x. Let S_x be an arbitrary k_x-sized set of child nodes z such that DecryptNode$(E, D, z) = F_z \neq \perp$, if S_x does not exists, DecryptNode$(E, D, x) = F_x = \perp$

$$\begin{aligned} F_x &= \prod_{z \in S_x} F_z^{\Delta_{i,S'_x}(0)}, \text{ where } {}_{S'_x = \{index(z): z \in S_x\}}^{i = index(z)} \\ &= \prod_{z \in S_x} \left(e(g,g)^{s \cdot p_z(0)}\right)^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z \in S_x} \left(e(g,g)^{s \cdot p_{parent(z)}(index(z))}\right)^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z \in S_x} e(g,g)^{s \cdot p_x(i) \cdot \Delta_{i,S'_x}(0)} \\ &= e(g,g)^{s \cdot p_x(0)} \end{aligned}$$

(ii) For each leaf node x

$$\text{DecryptNode}(E, D, x) = \begin{cases} \frac{e(D_i, E_x)}{e(D'_i, E'_x)} = \frac{e(g^r \cdot H(i)^{r_i}, g^{p_x(0)})}{e(g^{r_i}, H(i)^{p_x(0)})} = e(g,g)^{rp_x(0)} & \text{if } i = att(x) \in \gamma \\ \perp & \text{otherwise} \end{cases}$$

(iii) Apply this recursive algorithm to root node r, DecryptNode$(E, D, r) = e(g,g)^{rp_r(0)} = e(g,g)^{rs}$. Therefore

$$M = \frac{\tilde{E}}{(e(\dot{E}, \dot{D})/e(g,g)^{rs})} = \frac{\tilde{E}}{(e(h^s, g^{(\alpha+r)/\beta})/e(g,g)^{rs})}$$

In CP-ABE scheme, delegation of private keys means subtracting attribute from original attributes set γ to create a new more strict attributes set γ'. In the original paper, the delegation algorithm is accomplished by taking on unnecessary attributes from the original key and re-randomizing remaining attributes with new random numbers. This is much simpler than KP-ABE delegation. However, CP-ABE is more computational expensive than KP-ABE. That is because CP-ABE performs one more bilinear mapping for each leaf node than KP-ABE.

Chapter 4

ABE-based PHR System

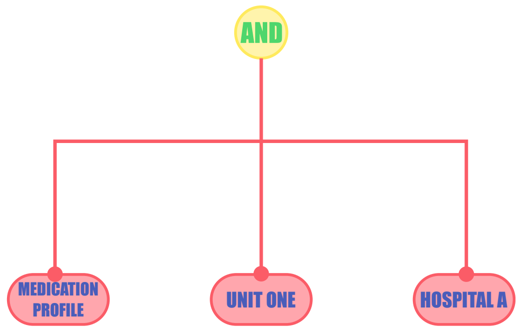
We use ABE as a building block of our proposed privacy-preserving PHR system. That is because ABE not only offers fine-grained access control similar to RBAC or ABAC, but also enforces data protection against semi-trusted server. However, several important issues arise when we try to create a PHR system based on ABE. Firstly, a PHR file may be operated by multiple users who the data owner may not know. It is hard to find a proper attributes universe $\mathcal{U}$ which can distinctly define each user. Improper attributes universe also complicates the access structure $\mathcal{A}$. Secondly, A trusted authority (TA) must be employed to protect the master key and generate private keys for the system users. Therefore, in this new scenario, there might be potential communication overhead related to key management (such as update and revocation). Thirdly, the only HBC secure encryption protocol for database, like PostgreSQL or MySQL, is client-side encryption (CSE). However, completely encrypting PHR file before submitting to the server will cripple certain features of PHR system, such as generating XML report in Indivo system. We propose a privacy-preserving PHR system model based on ABE. In order to justify our proposal, we create a Linux library (libcelia) and a toolkit (kpabe) implementing primitive KP-ABE. We also build an Indivo PHA which integrates the ABE-based security scheme into Indivo system. The performance of proposed model is being evaluated in two steps: First, we measure the time consumption of ABE using our toolkit and the CP-ABE toolkit created by John Bethencourt. Then, we measure the actual data query time when data is encrypted with ABE using our Indivo PHA.

4.1 Proposed PHR System Model

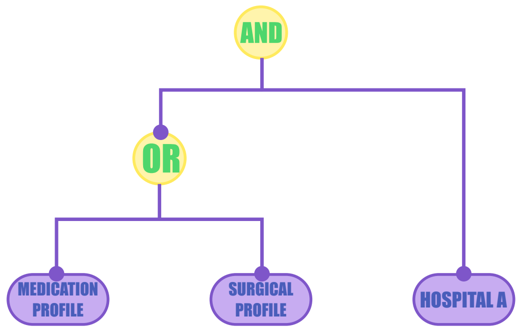
Attributes Set and Access Structure

Choosing a proper attributes set is the first step to build a efficient ABE-based PHR system. In our scheme, we divide the attributes universe $\mathcal{U}$ into two different attributes sets for KP-ABE and CP-ABE. For KP-ABE, we define a *date-based attributes set* using medical-specific attributes, such as surgical profile, medication profile, cardiac profile, etc. For CP-ABE we define a *role-based attribute set* using the role information such as john.doe.family, john.doe.friend and john.doe.physician. by doing so, we separate the attributes into two domains, namely the *public domain* which refers to the intrinsic properties of PHR data, and the *personal domain* which refers to the personal identifiable information of the entities in the PHR system. Nevertheless, we divide patients' PHR data in to two groups. One is data populated purely from professional EHR data. The other is data which patients aggregate themself. The later one can be originated from EHR data while allowing patients to add personal notations. Or, it can be data that patients create on their own, such as the daily blood sugar chart.

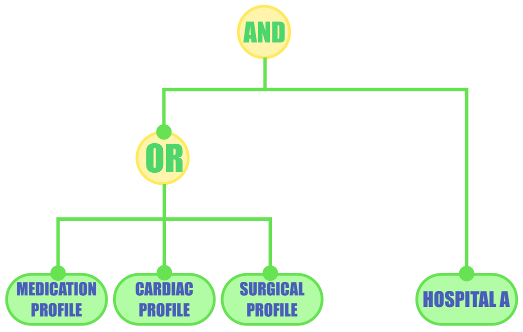
By separating the attributes into two non overlapping sets, we simplify the access structure $\mathcal{A}$ for both KP-ABE and CP-ABE schemes. For KP-ABE, each access structure $\mathcal{A}$ defines the privilege of individual medical personnel. For example,



(a) Access structure for nurse



(b) Access structure for surgical doctor

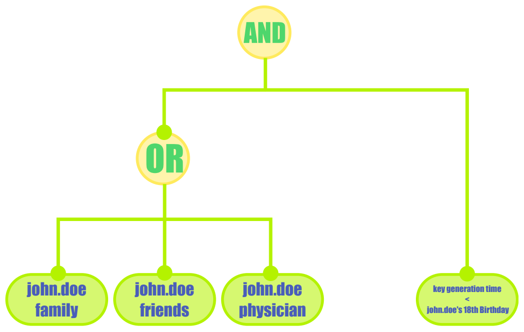


(c) Access structure for cardiac surgeon

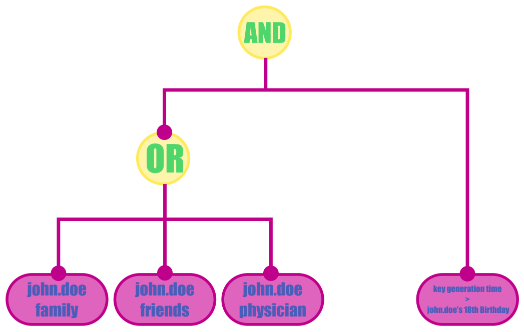
Figure 4.1: Access structures for different medical personnel

nurses may only be granted to access PHR data that labelled as medication profile. General surgical doctors may be granted to access PHR data that labelled as surgical profile or medication profile. Cardiac surgeons may be granted to access PHR data that labelled as surgical profile or medication profile or cardiac profile. Additionally, each medical personnel is usually "restricted by location". For example, nurses can only view the medical records of the patients assigned to the units they are on. Doctors can only view the medical records of the patients of the hospital they works. Therefore, the overall access structures for each medical personnel are shown in Figure 4.1. .

For CP-ABE, each access structure $\mathcal{A}$ determines who can access a particular PHR file. For example, a patient, John Doe, may allow either his family or his friends to access his weight chart. He may only allow his family to access his blood



(a) Access structure before 18th birthday



(b) Access structure after 18th birthday

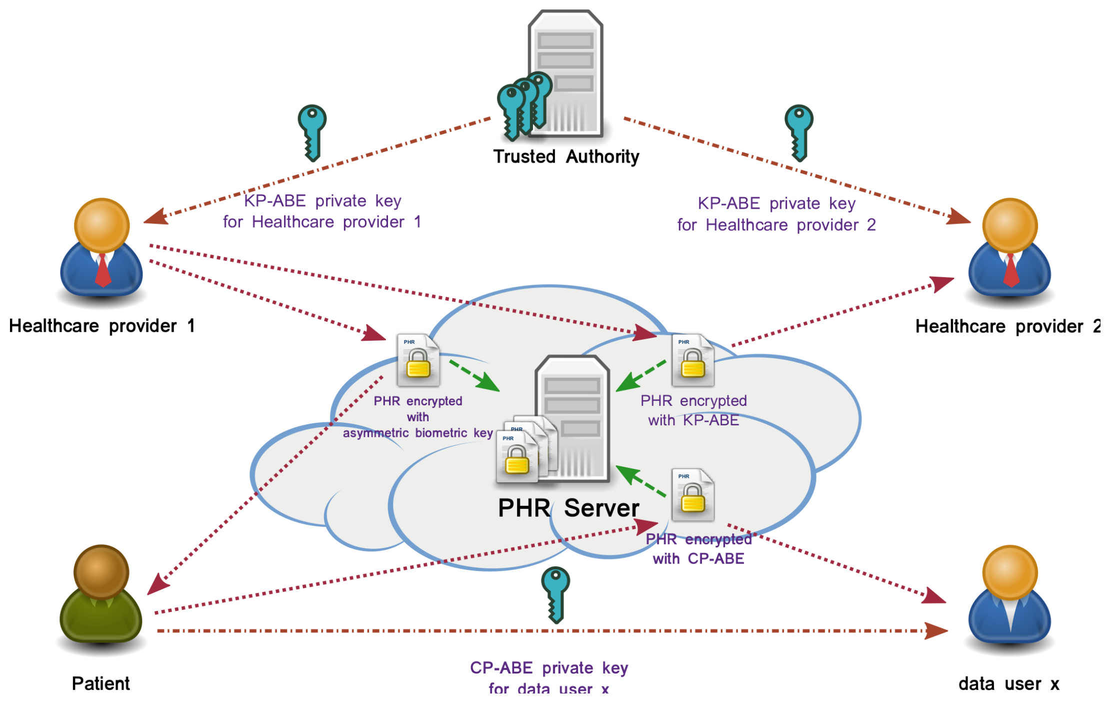
Figure 4.2: CP-ABE Access structure

sugar chart. Nevertheless, sometimes, he may even want to reserve some PHR files only for his doctor. In CP-ABE, we describe these access policies with role-based access structures. Patients may also include other attributes into *personal domain* to create more complicate access structures. For example, Parents or legal guardians of children may have privilege to access children's PHR when they are below certain age. However, once children reached certain age, for example at the age of eighteen, they should determine whether or not grant their parents further access to their PHR [ALG+]. In this case, we can include the data attribute into the attributes universe. Any private key should have a date attribute that describes the key generation time. As in Figure 4.5, any PHR file generated before the child's eighteenth birthday should require the key generation time to be less than the child's eighteenth birthday. While any PHR file generated after the child's eighteenth birthday should require the key generation time to be greater than the child's eighteenth birthday. Therefore, the new policy essentially revokes the old keys, which make it possible for children to control their parents' access rights.

System Architecture and System Operations

We would like to construct a PHR system that enhances the functionality and semantic interoperability of PHR data. Therefore, we consider the fact that not only patients but also healthcare providers will participate in aggregating PHR data. The overall system architecture is similar to an advanced E-Health Cloud model [LSW10]. Figure 4.1 illustrates our system architecture and the involved parties. There are four essential parties in our architecture: *patient*, *health care provider*, *trust authority*, *data user*. A *patient* is the person who own PHR data. He or she should be able to collect, manage and share his or her PHR data with other data users. A *health care provider* is a health professional or hospital which manages patients' electronic health records (EHR). Transitionally, patients' EHR are managed locally by health care providers. In our system, health care providers can export patients' EHR into patients' PHR repositories to ease the document transfer process when patients switch *health care providers.* A *trust authority (TA)* is usually an organization expected to be responsible for supervising healthcare information exchange among healthcare system. The regional health information organization (RHIO) is a typical TA. Since our system is build upon cryptography scheme, a TA here should also be responsible for distributing decryption keys to corresponding medical personnel. A *data user* is the person who is allowed to view a patient's PHR file. It can be a patient's family member, a patient's friend, a health insurance company, etc.

In the proposed PHR system, PHR server stores multiple copies of individual PHR file. Those copies can be encrypted under different cryptography schemes. The purpose of this setting is to preserve the veracity of the patients' PHR information. Consider the following scenario, Healthcare providers export patients medication profiles into the PHR server. Later, patients wants to make some annotations to



Trusted Authority
KP-ABE private key for Healthcare provider 1
KP-ABE private key for Healthcare provider 2
Healthcare provider 1
Healthcare provider 2
PHR encrypted with asymmetric biometric key
PHR encrypted with KP-ABE
PHR Server
PHR encrypted with CP-ABE
Patient
CP-ABE private key for data user x
data user x

Figure 4.3: Proposed PHR System Model. PHR server stores multiple copies of one PHR file encrypted with different cryptography schemes

those profiles to explain “their side of the story”. If there is only one copy of those profiles, any annotation made by patients will diminish the value of the original copies. Instead, we allow the patients to acquire other copies of those profiles from the healthcare provider. This time, patients are allowed to annotate, encrypt and share these extra copies. When patients switch to another healthcare provider, they can certainly share the annotated version with their new healthcare provider. Meanwhile, the new healthcare provider can refer to the original version to see what have been added by the patients themself.

The operations of proposed PHR system combine ABE and traditional cryptography, allowing patients to share their PHR data. These operations can be classified into following groups:

(i) **System Setup.** The TA, for example, a RHIO, invokes the $\mathsf{KPABESetup}(\mathcal{U})$

Table 4.1: Notation used in Proposed PHR System Model

Term	Definition
$\mathsf{KPABESetup}(\mathcal{U})$	Generate KP-ABE public key and master key using attribute universe $\mathcal{U}$
$\mathsf{KPABEKeyGen}(\mathcal{A}, MK)$	Generate KP-ABE secret key under access structure $\mathcal{A}$
$\mathsf{KPABEEncrypt}(\gamma, PK, m)$	KP-ABE encrypt m with KP-ABE public key and attribute set γ
$\mathsf{KPABEDecrypt}(SK, PK, CT)$	KP-ABE decrypt ciphertext CT with KP-ABE secret key ST
$\mathsf{CPABESetup}()$	Generate CP-ABE public key and master key
$\mathsf{CPABEKeyGen}(\gamma, MK)$	Generate CP-ABE secret key with attribute set γ
$\mathsf{CPABEEncrypt}(\mathcal{A}, PK, m)$	CP-ABE encrypt m with CP-ABE public key and access structure $\mathcal{A}$
$\mathsf{CPABEDecrypt}(SK, PK, CT)$	CP-ABE decrypt ciphertext CT with CP-ABE secret key ST
$\mathsf{ASKeyGen}(Bio)$	Generate asymmetric public key and secret key using biometric informations Bio
$\mathsf{ASKeyEncrypt}(PK, m)$	Encrypt m with asymmetric public key PK
$\mathsf{ASKeyDecrypt}(SK, m)$	Decrypt m with asymmetric secret key SK

function to generate a KP-ABE public key and master key. Each patient invokes the $\mathsf{CPABESetup}()$ function to generate his or her own CP-ABE public key and master key.

(ii) **ABE Key Distribution.** The TA generates KP-ABE private keys using $\mathsf{KPABEKeyGen}(\mathcal{A}, MK)$ with different access structures. Medical personnel obtain their KP-ABE private keys from TA (This should be a mandatory process similar to personnel registration). Patients generate CP-ABE private keys using $\mathsf{CPABEKeyGen}(\gamma, MK)$ with different role attributes. Patients then distribute these CP-ABE private keys to data users they intend to share PHR data with (This process should be done using a secure method such as emailing the private keys to data users using SSL).

(iii) **PHR Subscription.** Patients establish a "subscription" relationship with their healthcare providers. In this case, patients are required to physically go to their healthcare providers and provide proper identifications. The healthcare providers authenticate patients and generate asymmetric key pairs with $\mathsf{ASKeyGen}(Bio)$ using patients' biometric information. Additionally, patients may declare which part of EHR they wish to export into PHR server.

(iv) **Export EHR into PHR.** Healthcare providers identify new data or new change in EHR that need to be propagated to PHR. Then, healthcare providers encrypt a copy of these EHR data with $\mathsf{KPABEEncrypt}(\gamma, PK, m)$ and another copy with $\mathsf{ASKeyEncrypt}(PK, m)$. Both copies are exported into patients' PHR repositories.

(v) **PHR Management** patients can populate new PHR data by themself. Or, they can downloads the biometric asymmetric key encrypted version and decrypts it using $\mathsf{ASKeyDecrypt}(SK, m)$. In either case, patients are allowed to alter or annotate the decrypted PHR file. Patients encrypt these PHR data using $\mathsf{CPABEEncrypt}(\mathcal{A}, PK, m)$ with access structures as they thinks fit and submit them to the PHR server.

(vi) **Data User Access.** Data users are allowed to download those PHR data. Then, they may try to decrypt it using $\mathsf{CPABEDecrypt}(SK, PK, CT)$. If the attributes of data users' CP-ABE private keys satisfy the access structure of the ciphertext, they can successfully decrypt the PHR file, vise versa.

(vii) **Changing Healthcare Provider.** When patients change healthcare provider, the old healthcare provider updates certain attributes of those KP-ABE encrypted PHR data to allow new healthcare provider to access patients' PHR data using $\mathsf{KPABEDecrypt}(SK, PK, CT)$. Patients may also grant the new healthcare provider a CP-ABE secret key. The new healthcare provider can use this key to view the patients' altered version to gain a more comprehensive view of patients' health situations.

(viii) **Revocation** In both KP-ABE and CP-ABE scheme, If the master key holders wish to revoke a user's secret key, they need first identify the minimal attributes set that distinguish this user with others while minimizing affected group. Then, the following steps need to be carried out: (1) Update the corresponding attributes in the public key. (2) Update the corresponding attributes in the secret keys of all non revoked users. (3) Update the corresponding attributes in the ciphertext. For example, If a TA wishes to revoke a nurse's secret key. The TA identifies the *unit* attribute as the minimal set. That is because update *unit* attribute will only affect other nurses in the same unit. Therefore, the TA updates this attribute using the three steps above to revoke the nurse's secret key. Nevertheless, If patients wish to revoke friends' secret key. They identifies the friend attribute as the minimal set and update this component using the three steps above.

(ix) **Delegation** When data users wish to delegate part of their access rights to other data users, they eliminates certain attributes within their own secret keys to generate new secret keys will less attributes. The new secret keys are then passed on to other users. For example, data users have a secret key with both physician and friend attribute. They can pass on a new key with only friend attribute to other data users for access right delegation.

4.2 System Implementation

We prototype our system based on Indivo PCHR system by creating an Indivo PHA. The basic function of this PHA is helping doctors exporting their diagnostic notes to PHR server and viewing diagnostic reports generated by the PHR server. The security mechanism is enforced by applying a client side ABE to sensitive field within diagnostic notes before exporting them to the PHR server. Additionally, in order to support ABE operations, we create a Linux library (libcelia) and a toolkit (kpabe) implementing primitive KP-ABE.

4.2.1 Overview of the libcelia and kpabe

Library Structure

The Key-Policy Attribute-Based Encryption Library (libcelia) is a subroutine library implementing KP-ABE scheme. The source code of this library is divided into two part, core.c (Construction routines) and misc.c (Utility routines).

For this version, the library implements Setup, Encryption, Key Generation, and Decryption, Policy Generation algorithms. They are the most important parts during KP-ABE construction. For future release, Key Update and Key Delegation algorithms will be included since they are needed during PHR end-user application.

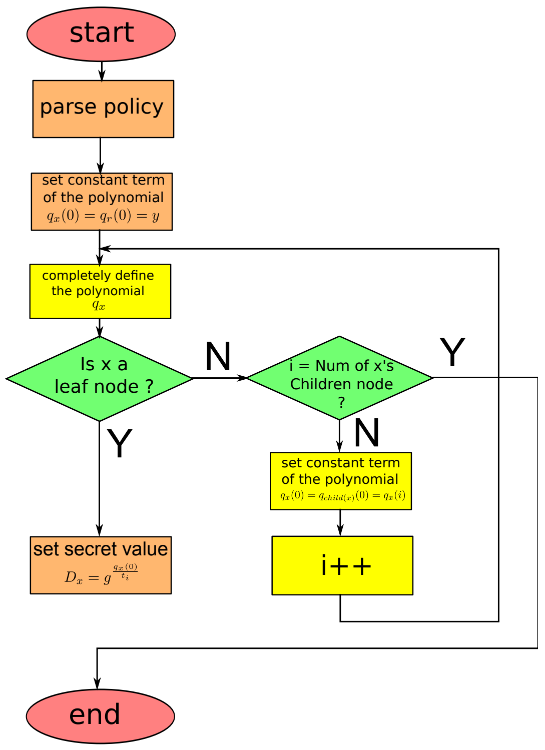
By default, the library uses *Type A Curves* during all pairing operations, since *Type A Curves* has the fastest pairing time compare to all other pairing types. However, using *Type A Curves* usually results in large group element size. By changing to *Type D Curves* in the source code, one can constrains the size of group element during pairing operation.

Essential Algorithms

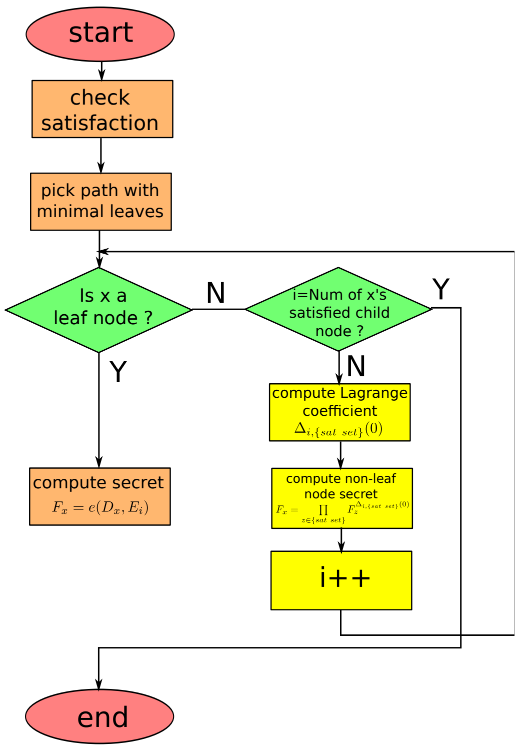
The essential algorithms during KP-ABE construction are Policy Generation algorithm (used during **Key Generation**), and Decryption algorithm.

The Policy Generation algorithm is implemented as a recursive algorithm. First, the function parse_policy_postfix parses the input policy string and fill out the k (threshold) and *attribute* element in the kpabe_policy_t structure. Then, function fill_policy implements the recursive algorithm by first setting the constant value of root polynomial $q_r(0)$ equal to the y element (PBC data type element_t) within kpabe_master_t structure. Then, it chooses *k-1* points on the finite field to completely define the polynomial. It also computes the constant values of its children node. Then it recursively invokes itself until current node becomes leaf node. If current node is leaf node, it computes the secret values D_x instead. Figure 4.4(a) shows the flow chart of this function.

The Decryption algorithm is implemented as three recursive algorithms. The first function check_sat is used to check whether the attributes of the ciphertext satisfy the access structure of the private key. It performs a bottom-up search. If there is a match between the leaf node attribute and a attribute within the ciphertext, the algorithm marks the *satisfiable* element in the kpabe_policy_t structure of that leaf node. If the number of satisfiable children nodes is larger than the threshold k of their parent node, the algorithm marks the *satisfiable* element in the kpabe_policy_t structure of the parent node. Therefore, upon finishing this algorithm, all the possible paths to the root node are marked. The second function pick_sat_min_leaves is mean to optimize the efficiency and find the path with minimal leaf nodes. It perform a top-down search by recursively rearranges satisfiable children nodes according to how many minimal leaf nodes they have. Finally, the third function dec computes DecryptNode(E, D, x) from the root node. If a node is leaf node, it decrypts the secret values with $e(D_x, E_i)$. Figure 4.4(b) shows the flow chart of these three functions.


start
parse policy
set constant term of the polynomial $q_x(0) = q_r(0) = y$
completely define the polynomial q_x
Is x a leaf node ?
N
Y
i = Num of x's Children node ?
Y
N
set constant term of the polynomial $q_x(0) = q_{child(x)}(0) = q_x(i)$
i++
set secret value $D_x = g^{\frac{q_x(0)}{t_i}}$
end

(a) Policy Generate Algorithm


start
check satisfaction
pick path with minimal leaves
Is x a leaf node ?
N
Y
i=Num of x's satisfied child node ?
Y
N
compute Lagrange coefficient $\Delta_{i,\{sat\ set\}}(0)$
compute non-leaf node secret $F_x = \prod_{z \in \{sat\ set\}} F_z^{\Delta_{i,\{sat\ set\}}(0)}$
i++
compute secret $F_x = e(D_x, E_i)$
end

(b) Decryption Algorithm

Figure 4.4: Essential Algorithms of KP-ABE

High level Functions and AES Encryption

The Toolbox (kpabe) is a set of programs implementing the high level functions of KP-ABE scheme. There are four programs in the toolbox, which are kpabe-setup, kpabe-keygen, kpabe-enc and kpabe-dec. Each program can be used as a standard shell command. Since pairing (bilinear mapping) is quite computationally expensive, we use AES symmetric key encryption to encrypt the actual plaintext and use KP-ABE to encrypt the symmetric key.

4.2.2 Using libcelia and kpabe

In this section we provide a brief tutorial on libcelia and kpabe. We first describe the process of building and installing the library, and then give some examples of how the library and toolbox are used in practice.

Building the Library and Toolbox

In order to build the library and toolbox, users need to have GNU multi-precision (GMP) library, pairing-based crypto (PBC) library, the development version of GNOME library (GLib), cryptogrphy and SSL/TLS toolkit (Openssl) installed first. The last two libraries usually can be installed directly from Linux distro repositories. The GNU multi-precision (GMP) library can be downloaded from `http://gmplib.org/` (both 4.3.2 version and 5.0.1 version works). The pairing-based crypto (PBC) library can be downloaded from `http://crypto.stanford.edu/pbc/download.html`. The installation processes are well documented on their respective websites.

If all required library are installed correctly. Users can start building the Key-Policy Attribute-Based Encryption Library. Both library and toolbox can be built with the standard GNU build system commands. Since the toolbox kpabe depends on the library libcelia. First, users should download, untar, compile, and install the most recent tarball of libcelia. The simplest way is run the standard GNU build system commands:

```
$ ./configure
$ make
$ make install
```

The "$" denotes your shell prompt.

The build process generates the static library `libcelia.a` which is be installed in /usr/local/lib directory.

After building and installing libcelia, users can build and install kpabe with the same commands.This time the it should generate four executable files: kpabe-setup, kpabe-keygen, kpabe-enc and kpabe-dec and corresponding manual files. The executable files are installed in /usr/local/bin directory.

Tutorial: Using Toolbox kpabe

Developer can directly include the head file celia.h into their source file to use KP-ABE subroutine functions. However, for a general user, it is easier to use the toolbox kpabe for KP-ABE operations. The synopses of these four shell commands are shown as follow:

(i) **kpabe-setup** Generate system parameters, a public key, and a master secret key under a set of attributes (ATTR) for using with **kpabe-keygen**, **kpabe-enc**, and **kpabe-dec**.

Syntax

```
kpabe-setup [-h] [-v] [-p FILE] [-m FILE] [-d] ATTR [ATTR ...]
```

Options

-h, –help	print the help message
-v, –version	print version information
-p, –output-public-key FILE	write public key to FILE
-m, –output-master-key FILE	write master secret key to FILE
-d, –deterministic	use deterministic “random” numbers (only for debugging)

(ii) **kpabe-enc** Encrypt FILE under a set of attributes (ATTR) using public key PUBLIC_KEY. The encrypted file will be written to FILE.kpabe unless the -o option is used. The original file will be removed.

Syntax

kpabe-enc [-h] [-v] [-k] [-o FILE] [-d] PUBLIC_KEY FILE ATTR [ATTR ...]

Options

-h, –help	print the help message
-v, –version	print version information
-k, –keep-input-file	don't delete original file
-o, –output FILE	write resulting ciphertext to FILE
-d, –deterministic	use deterministic "random" numbers (only for debugging)

(iii) **kpabe-keygen** Generate a key under the policy POLICY using public key PUBLIC_KEY and master secret key MASTER_KEY. Output will be written to the file priv_key unless the -o option is specified. If POLICY is not specified, the policy will be read from stdin.

Syntax

kpabe-keygen [-h] [-v] [-o FILE] [-d] PUBLIC_KEY MASTER_KEY [POLICY]

Options

-h, –help	print the help message
-v, –version	print version information
-o, –output FILE	write resulting ciphertext to FILE
-d, –deterministic	use deterministic "random" numbers (only for debugging)

(iv) **kpabe-keygen** Decrypt FILE using private key PRIVATE_KEY and assuming public key PUBLIC_KEY. If the name of FILE is X.kpabe, the decrypted file will be written as X and FILE will be removed. Otherwise the file FILE will be decrypted in place. Use of the -o option overrides this behavior.

Syntax

kpabe-dec [-h] [-v] [-k] [-o FILE] [-d] PUBLIC_KEY PRIVATE_KEY FILE

Options

-h, –help	print the help message
-v, –version	print version information
-k, –keep-input-file	don't delete original file
-o, –output FILE	write resulting ciphertext to FILE
-d, –deterministic	use deterministic "random" numbers (only for debugging)

The attribute parameter ATTR can be in two forms: *non-numerical* form and *numerical* form. *Non-numerical* attributes are simply any string of letters, digits, and underscores beginning with a letter. *Numerical* attributes are specified as "attr = N", where N is a non-negative integer less than 2^{64} and "attr" is another string. The whitespace around the "=" is optional. One may specify an explicit length of k bits for the integer by giving "attr = N#k". Note that any comparisons in a policy must then specify with the same number of bits. Since **kpabe-setup** need every possible attributes to be include in the attributes universe. *Numerical* attributes for **kpabe-setup** are specified as "attr = " or "attr = #k" with no specific number "N".

The policy parameter POLICY is string that concatenates ATTR with key words, such as "and", "or", "of". Nevertheless, in POLICY, *Non-numerical* attributes can be specified as "attr $<$ N", "attr $\leq$ N", "attr $>$ N", "attr $\geq$ N" and "attr = N".

We can better demonstrate these features with the example in Figure 4.2.2. In this example, the shell backslash "\" represents line continuation. The private key generated in this example can only decrypt ciphertext whose severe level is less than 4 and being labelled with at least two attributes from medication profile, surgical

```
$ kpabe-setup medication_profile surgical_profile cardiac_profile \
>'severe level = '
$ kpabe-enc pub_key PHR.pdf medication_profile surgical_profile \
>scardiac_profile, 'severe level = 3'
$ kpabe-keygen -o doctor_priv_key pub_key master_key \
>'severe level < 4 and 2 of (medication_profile, surgical_profile, cardiac_profile)'
$ kpabe-dec pub_key doctor_priv_key PHR.pdf.kpabe
```

Figure 4.5: Sample usage session of kpabe

profile, and cardiac profile. In this case, this private key is sufficient to decrypt the encrypted file PHR.pdf.kpabe.

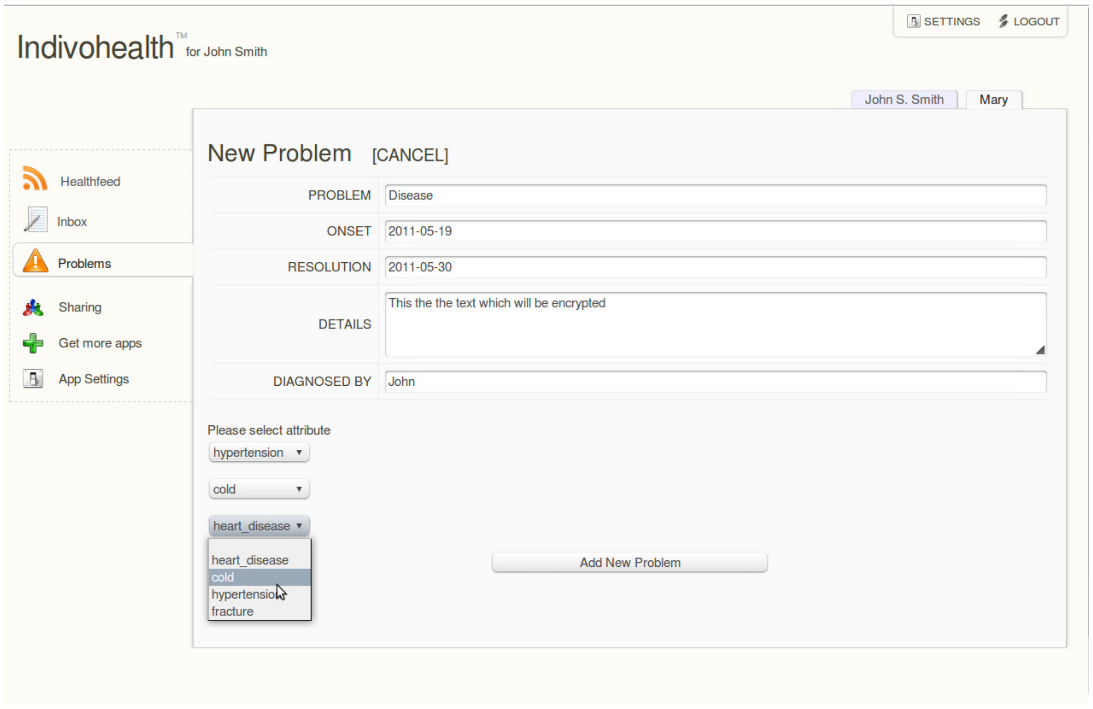
4.2.3 Indivo PHA Implementation

For the purposes of testing our PHR system architecture, we implemented an Indivo PHA based on the indivo X PCHR system. This python-based PHA allows doctors to create diagnosis notes and submit them to the patients' PHR repositories.

If patients wish to authorize their doctor to export diagnostic result to their' PHR repositories, They can define the sharing groups when authenticate this PHA. The PHA completes an OAuth authorization process with users' indivo_record_id and other necessary informational content. All members within the sharing groups will be able to access the patients' PHR record through this PHA.

The PHA is integrated with ABE functionality to achieve fine-grained access control. When doctors export their diagnostic results, they can select several attributes to define the diagnostic results, as shown in Figure 4.2.3. The PHA will encrypt the input text with these attributes using ABE. Then, it fills the encrypted text to a XML template and submits it to the PHR backend server.

When data users wish to access these diagnostic notes, the PHA will request a diagnostic summary from the backend server. The backend server generates a



New Problem [CANCEL]

PROBLEM	Disease
ONSET	2011-05-19
RESOLUTION	2011-05-30
DETAILS	This the the text which will be encrypted
DIAGNOSED BY	John

Please select attribute

hypertension

cold

heart_disease

Figure 4.6: Select attribute to encrypt the diagnostic result

XML report listing all diagnostic reports within patients' PHR repositories. Data users can further request to access a specific diagnostic report. The PHA will ask data users to submit their ABE private key and download the requested diagnostic report. If the ABE private key can decrypt the corresponding file, the PHA will show data users the actual content of the report. Otherwise, it will alert the data users they do not have proper access right.

4.3 System Performance Evaluation

We evaluate the performance of purposed PHR system in two steps: First, we measure the computation time of ABE using kpabe toolkit, and the cpabe toolkit created by John Bethencourt. Next, we measure the data query time of our Indivo PHA in different settings. All measurements are taken on a workstation with 3.4

Table 4.2: PBC Library Benchmark

Curve Type	Average Pairing times (ms)	Base field size (bits)	Average Pairing times (preprocessed) (ms)
A	512	5.58	2.465
D159	159	13.9	10.647
D201	201	19.877	14.808
D224	224	24.198	17.651
E	1024	21.662	21.888
F	160	67.140	67.877
G149	149	40.279	36.792
A1	1024	146.176	37.471

Ghz Pentium duo core 32-bit processor.

4.4 ABE Performance Measurements

Both kpabe and cpabe toolkit use a 160-bit A type elliptic curve group based on the supersingular curve $y^2 = x^3 + x$ over a 512-bit finite field without preprocessing. Since the most time consuming computation in ABE operations is computing bilinear map. We first give a benchmark of PBC library on the test machine (Table 4.2). Roughly speaking, the overall decryption time is in direct proportion with PBC library benchmark. Therefore, we can estimate the decryption time when using other type of elliptic curve groups.

The other two significant operations are exponentiation and randomly selecting element. In our setting, a exponentiation consume 6.0ms in an elliptic curve group and 0.56ms in a finite field. Randomly selecting element is accomplished by calling the random function in general purpose standard library. The random function generate a random number by reading from the random number generator in the Linux kernel /dev/urandom. The overall time consumption is 15ms in an elliptic curve group and 1.5ms in a finite field.

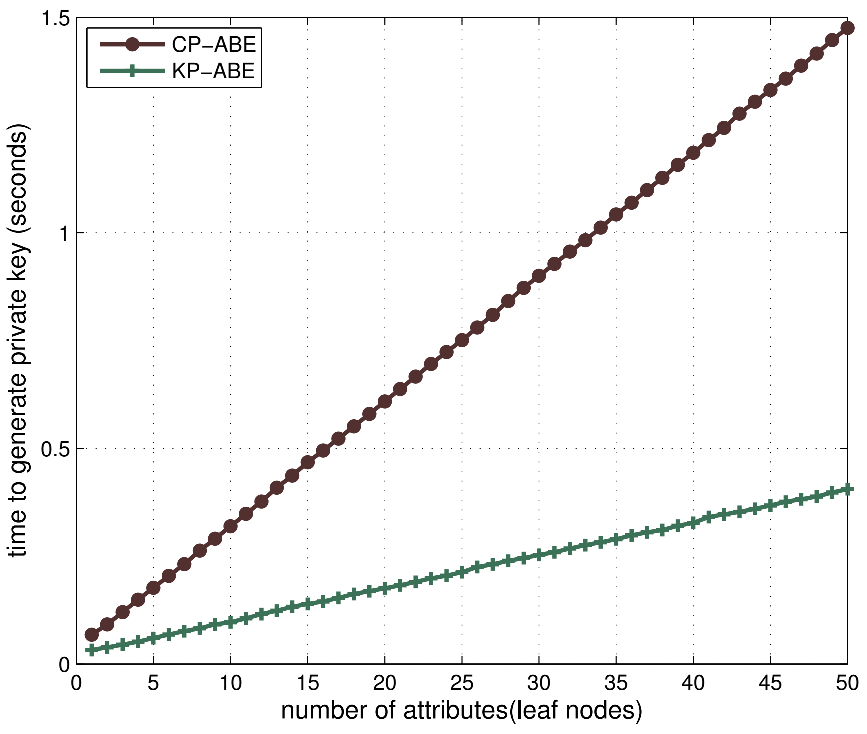


Figure 4.7: ABE Key Generation Time

In our test, we define the access structure as a simple AND gate. Therefore, there is no internal node within the access structure. As expected, the Key generation time of ABE, shown in Figure 4.4, is precisely linear in the number of attributes (in CP-ABE) or leaf nodes (in KP-ABE). For CP-ABE, the key generation process needs to select one random element in $\mathbb{Z}_p$ (a finite field) and perform three exponentiations in G (an elliptic curve group) for each attribute. For KP-ABE, the process only needs to select one random element in $\mathbb{Z}_p$ (during the construction of polynomial p) and perform one exponentiation in G for each leaf node. Therefore, the key generation time of KP-ABE is roughly three times faster than the key generation time of CP-ABE.

For encryption, shown in Figure 4.4, CP-ABE encryption algorithm needs to to select one random element in $\mathbb{Z}_p$ (during the construction of polynomial p) and

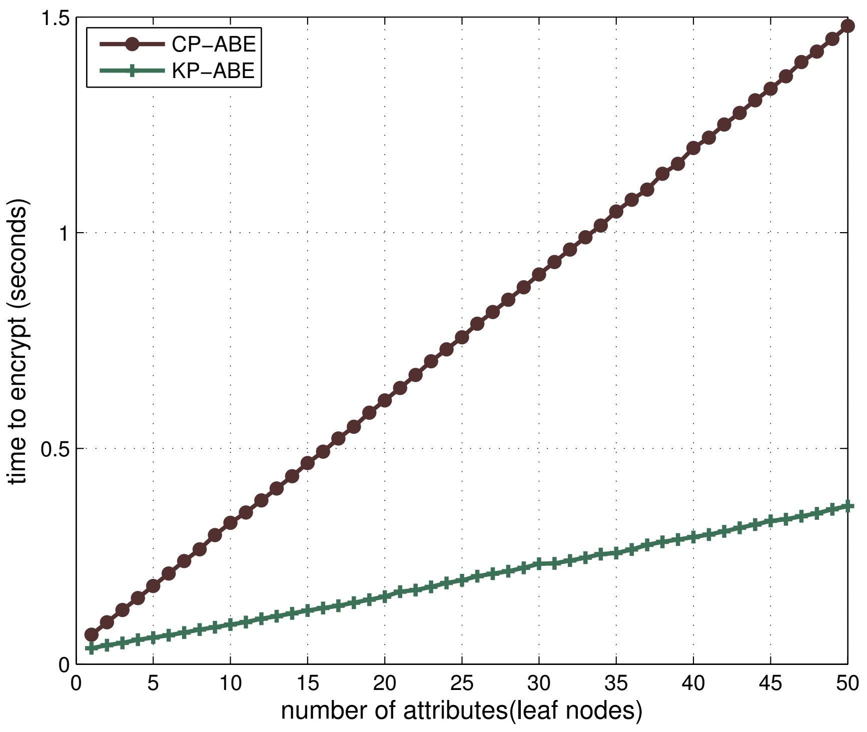


Figure 4.8: ABE Encryption Time

perform two exponentiations in G (an elliptic curve group) for each attribute. While, KP-ABE encryption algorithm only needs to perform two exponentiations in G (an elliptic curve group) for each attribute and select only one random element in $\mathbb{Z}_p$ for all attributes. Therefore, the encryption time of KP-ABE is still roughly three times faster than the encryption time of CP-ABE.

As for Decryption, the most expensive computation is bilinear mapping which CP-ABE does twice and KP-ABE only does once for each leaf node. Therefore, the decryption time of KP-ABE is roughly twice faster than the decryption time of CP-ABE, as shown in Figure 4.4.

Though, KP-ABE is more computationally efficient than CP-ABE in all three aspects, There are two noticeable disadvantages when using KP-ABE, First of all, the sizes of public key and master key grow linearly with the number of attributes in

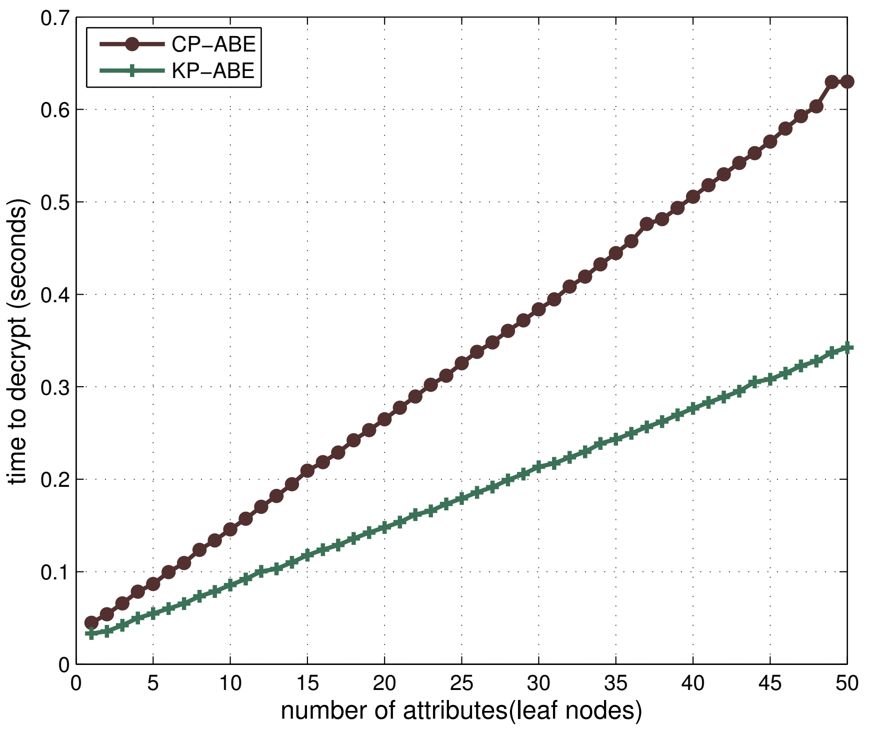


Figure 4.9: ABE Decryption Time

the universe. This could cause communication overhead for the system, especially when system frequently requires users to update their public key. Figure 4.4 show relation between the sizes of public key and mater key and attributes in the universe. Noticing that a *Numerical* attribute is represent by up to 64 distinct attributes in our algorithm. The actual KP-ABE attributes universe of a PHR system can be fairly large. However, we can better constrain key size by using large universe construction of KP-ABE.

Second of all, as in Figure 4.4, during setup phase, KP-ABE requires significantly longer time to generate public key and master key than CP-ABE when attributes universe increases. That is because KP-ABE setup process needs to select one random element in $\mathbb{Z}_p$ (a finite field) and perform one exponentiations in G (an elliptic curve group) for each attribute. The CP-ABE setup time, on the other

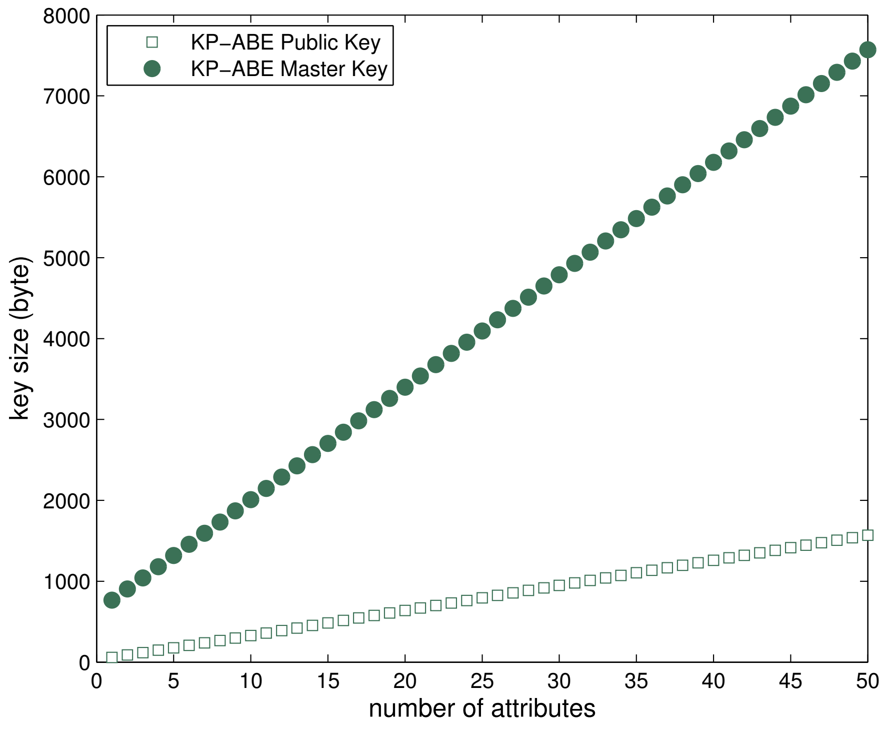


Figure 4.10: KP-ABE Key Size

hand, remains constant (approximately 60 ms), since it does not perform additional operations when attributes universe increases.

4.5 Indivo PHA Data Query Measurements

In order to better evaluate the performance of a crypto-based PHR system. We measure the data query time of Indivo system using our PHA. The testing scenario is loosely based on TPC-B, involving five SELECT, UPDATE, and INSERT commands per transaction. The data involved is encrypted and decrypted using the symmetric key encryption and decryption functions provided by the pgcrypto module in PostgreSQL. The symmetric key is randomly generate by our PHA and encrypted using ABE. For example, the SELECT and UPDATE commands is exe-

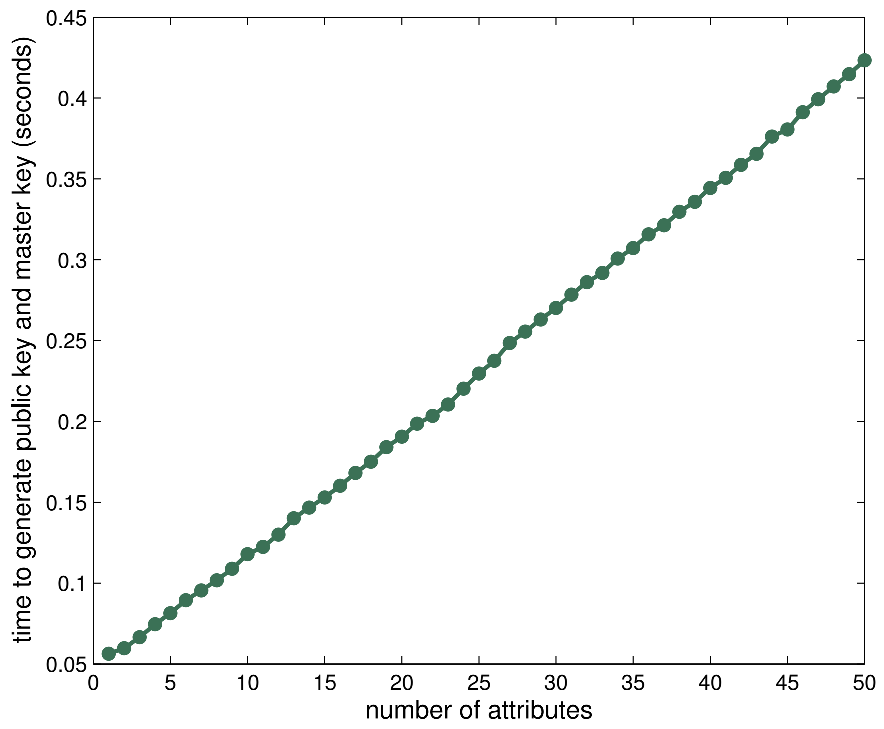


Figure 4.11: KP-ABE Key Size

cuted as follow:

```
UPDATE pgbench_accounts SET abalance = pgp_sym_encrypt('secret', :psw,
'compress-algo=1, cipher-algo=aes256') WHERE aid = :aid;
SELECT pgp_sym_decrypt(abalance, :psw, 'compress-algo=1, cipher-algo=aes256')
FROM pgbench_accounts WHERE aid = :aid;
```

Where the "psw" is the decrypted password from a ABE ciphertext. We test the transaction script under different circumstances. First, we test it under different database size. As shown in Figure 4.5, the actual database size (number of rows) is equal to $100000 *$ scale factor. In Figure 4.5, the data query time is almost linear to the number of attributes. The size of the database has negligible effect to the

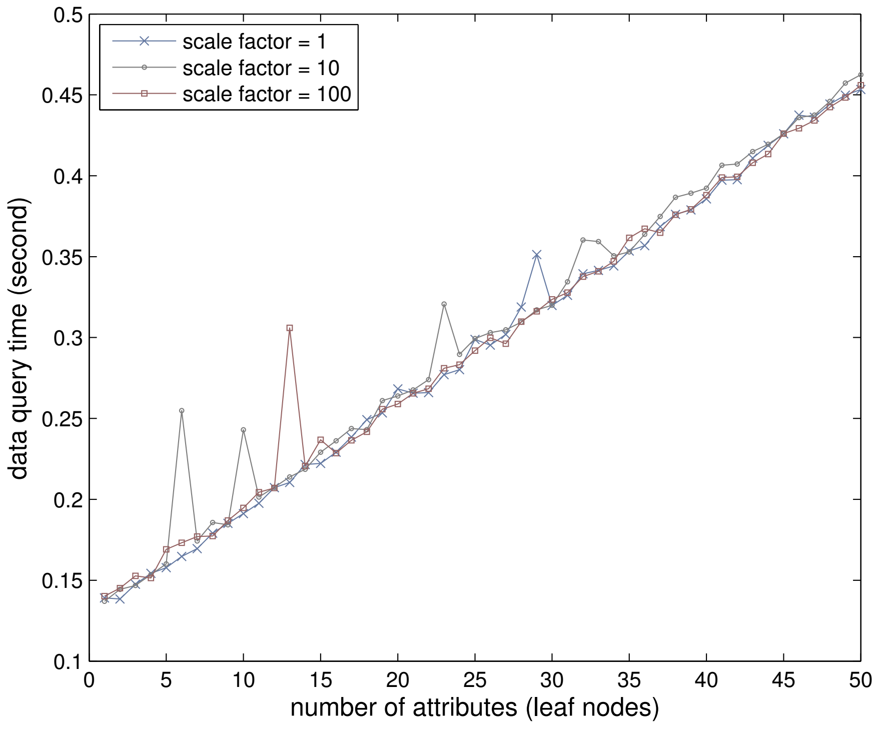


Figure 4.12: Data Query Time with Different Database Sizes

final data query time. That is because the time consumption of ABE operation dominates the overall data query time. Compare to ABE operation, a normal TPC-B transaction usually takes less than 1.6 ms on our test machine.

The data query time under multiple data users is somewhat more interesting. In Figure 4.5. data query time in multiple clients setting is also linear to the number of attributes. However, not like in single client setting. if we enlarge the figure, shown in Figure 4.13(b), we can see the number of clients does affect the overall data query time. Imaging a PHR system deployed nationally, the number of clients who access this system simultaneously could raise to hundred of thousands. Such circumstance can noticeable query time delay for the system.

These measurements give some insight into the effect of ABE operation within a PHR system. In both cases, the main portion of data query time is consumed by

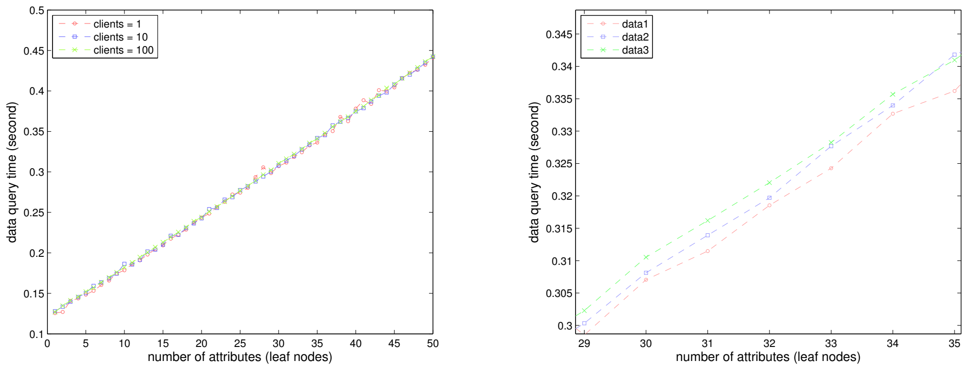


Figure 4.13: Data Query Time with Different Number of Clients

cryptography operations. Consider the fact that an average of 1000 tps (transaction per second) is usually the benchmark of commercial database, we believe further optimization must be employed to integrate ABE into PHR system.

Chapter 5

Conclusions and Future Work

In this thesis, we address the security and privacy concerns of cloud-based PHR system by integrating advanced cryptographic techniques, such as ABE, into PHR system. We demonstrate that, by using appropriate cryptographic techniques, patients can protect their valuable healthcare information against partially trustworthy cloud server. Meanwhile patients gain full control over access to their PHR files, by assigning fine-grained, attribute-based access privileges to selected data users. We study the possibility of extending PHR interoperability by importing professional electronic health record (EHR) into PHR system using ABE.

We briefly reiterate our main original contributions:

(i) We proposed a secure PHR system framework using ABE to realize fine-grained, patient-centric access control. We manage to simplify the access structure by dividing users into two domains. We use different ABE scheme to share PHR data with users from different domains.

(ii) To our best knowledge, we created the first Linux library that implement primitive of key policy attribute-based encryption (KP-ABE) algorithms.

(iii) We implemented an Indivo PHA application that allow doctors to encrypt and submit their prescription and diagnostic notes using KP-ABE.

(iv) We evaluate the performance efficiency of different ABE schemes as well as the data query time of Indivo PCHR system when PHR data are encrypted under ABE scheme

Upon finishing our system design, we realized cryptographic techniques represent only one pillar in an overall approach to effective and efficient PHR security. Future work of this project might include:

(i) Enrich ABE library. There are several limitations and shortages in our ABE library. The time cost of the ABE library significantly reduces the security benefit. Therefore, we will try to include new ABE construction into our ABE library to optimize its performance.

(ii) Explore more dynamic and efficient way to illustrate access structures. Currently, we describe access structures in a static way, which is not suitable for a dynamic system like PHR. A better description method can further reduce key management complexity and policy complexity.

(iii) Combine other privacy-enhancing techniques with cryptographic techniques. Cryptographic techniques is an essential, but not the only one, method to protect private data against partially trustworthy cloud server. Therefore, we are trying to find more efficient way to address the security and privacy issue of PHR systems.

Bibliography

[ALG$^+$] J.A. Akinyele, C.U. Lehmann, M.D. Green, M.W. Pagano, Z.N.J. Peterson, and A.D. Rubin. Self-protecting electronic medical records using attribute-based encryption. Technical report, Cryptology ePrint Archive, Report 2010/565, 2010. http://eprint. iacr. org/2010/565.

[ASZ$^+$10] B. Adida, A. Sanyal, S. Zabak, I.S. Kohane, and K.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.

[Bei96] A. Beimel. Secure schemes for secret sharing and key distribution. *DSc dissertation*, 1996.

[BSW07] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. 2007.

[GPSW06] V. Goyal, O. Pandey, A. Sahai, and B. 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.

[Kau09] L.M. Kaufman. Data security in the world of cloud computing. *Security & Privacy, IEEE*, 7(4):61–64, 2009.

[KJJ$^+$08] D.C. Kaelber, A.K. Jha, D. Johnston, B. Middleton, and D.W. Bates. A research agenda for personal health records (phrs). *Journal of the American Medical Informatics Association*, 15(6):729–736, 2008.

[KPB06] Anders Kofod-Petersen and Terje Brasethvik, editors. *Electronic Personal Health Records, Proceedings of the International Symposium on Electronic Personal Health Records 2006 (ISePHR 2006)*, volume 219, Trondheim, Norway, September 2006. CEUR Workshop Proceedings.

[LSW10] H. L
"ohr, A.R. Sadeghi, and M. Winandy. Securing the e-health cloud. In *Proceedings of the 1st ACM International Health Informatics Symposium*, pages 220–229. ACM, 2010.

[Lyn07] B. Lynn. *On the implementation of pairing-based cryptosystems*. PhD thesis, Citeseer, 2007.

[LYRL10] M. Li, S. Yu, K. Ren, and W. Lou. Securing personal health records in cloud computing: Patient-centric and fine-grained data access control in multi-owner settings. *Security and Privacy in Communication Networks*, pages 89–106, 2010.

[MSCA07] K.D. Mandl, W.W. Simons, W.C.R. Crawford, and J.M. Abbett. Indivo: a personally controlled health record for health information exchange and communication. *BMC medical informatics and decision making*, 7(1):25, 2007.

[SMK05] W.W. Simons, K.D. Mandl, and I.S. Kohane. The ping personally controlled electronic medical record system: technical architecture. *Journal of the American Medical Informatics Association*, 12(1):47, 2005.

[SW05] A. Sahai and B. Waters. Fuzzy identity-based encryption. *Advances in Cryptology–EUROCRYPT 2005*, pages 457–473, 2005.