4-1-2013

Toward privacy-preserving emergency access in EHR systems with data auditing

Muhannad Darnasser

Toward Privacy-Preserving Emergency Access in EHR Systems with Data Auditing

by

Muhannad Darnasser

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
Master of Science
in
Computer Science

Supervised by

Dr. Rajendra K. Raj

Department of Computer Science

B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

April 2013

The thesis “Toward Privacy-Preserving Emergency Access in EHR Systems with Data Auditing” by Muhannad Darnasser has been examined and approved by the following Examination Committee:

Dedication

I lovingly dedicate this thesis to my precious fiance, who encouraged and supported me with patience and love each and every step of the way.

Acknowledgments

I would like to thank Prof. Raj for his support, encouragement, and advice which without, this work would have never seen the light. I would also like to thank Suhair for being a great reader and for her support, and Prof. Radziszowski for his support and advice.

I would also like to thank Fulbright and Amideast for granting me this opportunity and for all their hard workers who made this journey unforgettable.

At the end, I am grateful for my family, mom and dad, my fiance, and my friends for all the support, encouragement, and just for being there when needed.

THANK YOU ALL.

Abstract

Toward Privacy-Preserving Emergency Access in EHR Systems with Data Auditing

Muhannad Darnasser

Supervising Professor: Dr. Rajendra K. Raj

Widespread adoption of health information sharing is claimed to improve healthcare quality at reduced cost due to the ability for providers to share healthcare information rapidly, reliably, and securely. During emergency access, however, such sharing may affect patient privacy adversely and steps must be taken to ensure privacy is preserved. Australia and the US have taken different approaches toward health information sharing. The Australian approach broadly uses a "push" model where a summary record is extracted from local health records, and pushed into a centralized system accessed by providers. Under the US approach, providers during emergency access generally "pull" health records from a centralized system that typically replicates local health records. On the other hand, the centralized repository most likely will be a third party cloud provider that offers on demand availability of high quality and cost effective services. These features make cloud computing a perfect infrastructure for EHR systems. The fact that medical data are handled and managed by a third party cloud provider, however, requires additional security mechanisms, i.e. auditing, to preserve data confidentiality, integrity, and privacy. This thesis contrasts the Australian and US approaches to information sharing during emergency access, focusing on patient privacy preservation. It develops a generalized approach to enhance patient privacy during emergency access using “push” and “pull” approaches. It presents an auditing service implementation over a multi-cloud data repository. It finally shows preliminary results from a proof-of-concept EHR system.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

The Health Information Technology for Economic and Clinical Health (HITECH) Act supports the concepts of Electronic Health Record (EHR) and Health Information Exchange (HIE). Healthcare providers and organizations that show “meaningful use” of these technologies receive incentive payments, thus helped to increase usage of EHR systems in the U.S. among providers [26]. By 2012, almost 48% providers were e-prescribing on an e-prescription network. EHR systems play a major role in health data sharing, which has shown to improve the quality of healthcare services, increase efficiencies, provide cost savings, and most importantly, help save lives in emergency situations by permitting healthcare providers deliver the best emergency care in the shortest time.

Healthcare providers are permitted under the Health Insurance Portability and Accountability Act (HIPAA) Privacy Rule [23] to access critical patient data without prior consent in emergencies, but only for the purposes of patient treatment. Accessing such medical data electronically allows providers to deliver an efficient emergency treatment. Consider a situation where Bob got into an accident and reached an emergency center in a nearby hospital unconscious. The emergency room doctor cannot provide an effective treatment without knowing Bob’s current medications, any allergies, and other medical problems, e.g., diabetes. With the availability of emergency care systems, the doctor can use patient identification information to access any required information immediately and provide the needed treatment. Without such information at hand, the doctor would first need to get the required tests performed before providing treatment. In other words, having access to EHR data during emergencies saves time, reduces errors, and results in improving healthcare.

Despite these benefits, accessing health data during emergency situations faces challenges in preserving patient privacy due to the reduced impact of privacy regulations in such situations. Healthcare organizations face internal and external threats, however, internal threats have been the most common [20]. In the US, medical data breaches costs $234 billion annually estimated by FBI [27]. Emergency access represents one of the easiest methods to access unauthorized data because the hacker needs only to provide a plausible reason for access. Since patient data contains not only medical related data but often management, financial, and personal data, the damage due to such breaches can be huge for both the patient and organization. A common fraud is medical identity theft where "a person uses someone else's medical record to obtain medical goods or services or to bill for medical goods and services that the patient did not receive" [27]. This kind of fraud often occurs by exposing non-medical data. For example, an incident where 2252 medical records were accessed improperly by one or more employees when patients visited emergency departments at three Central Florida hospitals between January 1, 2010 and August 15, 2011. Non-medical data such as patient names, social security numbers, dates of birth, and insurance information were exposed [1].

Approaches toward the implementation of EHR systems varies between giving healthcare providers the ability to control which health data is extracted and "*pushed*" into a central health data repository, or replicating local health data, that are stored and maintained in healthcare providers offices, into a central repository to be "*pulled*" by other healthcare providers. Generally, Australia adopts the first model while the U.S. follows the second approach [15].

Our system focuses on separating emergency data from the core of EHR systems, in order to minimize the amount of leaked data in cases of emergency access breaches. Also this separation removes the burden of building an emergency access system in the core of the EHR system. Since enabling emergency access in EHR system may force the system to compromise some security measures and introduce security holes, in order to give the users the ability to override access policies and security rules in the system to gain access to medical data when needed. By following our model we can first build an efficient EHR system that focuses on preserving privacy and confidentiality with the ability to provide full data records for normal daily usage. Then we can build a separate efficient emergency system that focuses on delivering the needed data without any delay and preserving patient's privacy and confidentiality.

On the other hand, for an EHR system to be able to cover large areas and provide services to customers all around the US, EHR system must be able to handle all requests from patients and providers in a reasonable time, provide data replication and synchronization for availability, provide services in crisis times, handle the huge data for all patients, auto scale the resources to handle bottleneck times, and be fairly cheap. All those characteristics and more are found in a database as a service in the cloud [6]. But by moving data to the cloud, the data owners are relinquishing their complete control over their data and giving private information to an unauthorized users. Handling medical data by a third party requires extra measures by the data owners such as encryption to preserve privacy and confidentiality which is covered by legal regulations such as HIPAA [23]. In addition to the lack of authority over the data stored under their control, there are no guaranties that the cloud providers will not act according to their benefits and needs to delete rarely used data, tamper with the data, or hiding data loss incidents to preserve their reputations. For examples [5] and [13] are incidents where data has been lost in two majors cloud providers Google and Amazon.

As a result of those threats and the need to preserve patients privacy, confidentiality, and integrity, the system needs to provide a mechanism to prove data integrity and correctness while being stored in the cloud, which can be provided by data auditing in addition to data encryption. Our system will introduce a mechanism to audit the data while being stored in the cloud and will give the users the ability to verify their data integrity and correctness without the need of introducing a third party auditor like [30] or revealing any private data to the cloud provider.

1.2 Background

1.2.1 Cloud Computing

"Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data-centers that provide those services" [6]. The cloud is divided into three major categories, Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), also it can be extended to add Database as a Service (DaaS). The cloud offers a scalable and distributed storage system that can help develop large scale applications to handle large scale databases and cover large areas [2]. The cloud providers offer services that dynamically vary according to workload (Elasticity), charge the customers only for what they use (pay-per-use), low upfront investment, low time to market, and transfer the risks from customers to the cloud. The customers only store their data in the cloud and the providers distribute the data into different data centers and guarantee an acceptable response time for customers with minimum downtime. Comparing cloud providers with private storage systems that are managed by local administration, cloud providers have dedicated hardware to provide the best service, different kinds of security measures to defend against external attacks, up-to-date software, and dedicated experts in every field to cover all the needs for the cloud providers. All those benefits and more are provider to customers with very reasonable costs, which in most cases cannot be covered by private businesses because of the high costs and complexity of building and maintaining such dedicated systems.

While these advantages make the cloud a perfect infrastructure for large and small applications, the cloud brings new security risks toward outsourced data. The fact that the cloud provider is a third party storage system that does not have the authority to view or edit the data is a risk by itself. Also the interests of the data owner may collide with the interests of the cloud provider that may lead to harming the data itself. On the other hand, cloud providers may also hide some incidents from the customers to preserve their reputation and prevent customers from raising law suits against them. All the previous risks are internal risks, but the cloud also introduces new external risks such as the fact that all different kinds of data are collected in one place which will be an attractive target for hackers much more than local small storage systems. Public cloud auditing [30] and data encryption [18] can be used to protect the data against internal and external risks.

1.2.2 Encryption schemes

Encrypting data is the first and essential line of defense against attackers and unauthorized users. Data will be encrypted while being stored in the cloud to prevent cloud providers from accessing or tampering the data. Also data will be encrypted while being transferred between entities in the system or while being stored in some temporary locations such as doctors local machines. The following sections describes the various encryption schemes and their usage.

Symmetrical Encryption

Symmetric-key encryption schemes are schemes in which the encryption key and the decryption key are the same. One of the schemes that follows symmetric-key encryption is Advanced Encryption Standard (AES) [29]. In those schemes the user must share the key with all the users who have the authority to view the encrypted data and if the user needs to give different access permissions to different users then there is a need to encrypt each part of the data with different keys and share each key with the authorized users only.

Therefore, in a group of users each two users must have a unique key shared between them for secured communication. Symmetric-key encryption systems consist of multiple rounds of substitution-permutation networks, that is why it is fast and simple, but with a large key and a good number of rounds, the system becomes secure and hard to break. However, the system becomes too complicated and hard to manage when the number of users increases and with the need of fine-grained access policies because of the huge number of keys and the work needed by the users and the system to manage data access [8]. Symmetric-key encryption will be used in the system to encrypt emergency report while being stored in the cloud.

Public-Private Encryption

Public-Private encryption schemes are schemes that use two different keys for data encryption and decryption. The user first generates a private key that should be kept secured, and a public key which is released to the users. Any data that was encrypted by the public key can only be decrypted by the private key and vice versa. RSA [28] is a famous public-private encryption scheme. So if Alice wants to share data with Bob, she just needs to know the public key for Bob and use it to encrypt the message. When Bob receives the encrypted message he will use his private key to decrypt the message and read its contents. Therefore, in a group of users, each user must have one pair of keys, a private one that is hidden from the others, and a public key that is shared with all the users. The users in those schemes should only manage their private keys, while the public keys can be kept in a shared place.

CP-ABE

Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [9] is a public key encryption scheme that enables the user to specify who can access the data when encrypting it by constructing a fine-grained access policy and associate the policy with the data. Each user in the system must have a private key associated with a list of attributes that describes the user, the user can decrypt the data file if and only if her key's attributes can satisfy the access policy that was associated with the encrypted data. CP-ABE consists of a master key, a public key, a set of attributes, a set of private keys, and four fundamental algorithms (Setup, Encrypt, Key Generation, Decrypt) in addition to one optional algorithm (Delegate).

- **Setup:** it only takes an implicit security parameter to generate both the Master Key (MK) which is kept hidden from all users and the Public Key (PK) which is shared with all users.
- **Key Generation(MK,S):** generates a user private key (SK) using the master key *MK* and a set of attributes that describes the owner of the key *S*.
- **Encrypt(PK,M,$\mathbb{A}$):** generates ciphertext (CT) for the message *M* using the public key *PK* and associate it with a user defined access policy $\mathbb{A}$ such that only users with keys that their attributes satisfy $\mathbb{A}$ can decrypt the ciphertext *CT*.
- **Decrypt(PK,CT,SK):** decrypts the ciphertext *CT* in association with the public key *PK* and the users private key *SK* and generates the original message *M* if and only if the the set of attributes *S* that is associated with the private key *SK* satisfies the access policy $\mathbb{A}$ that was associated with the ciphertext *CT*.
- **Delegate(SK,S'):** generates another user private key *SK'* from another private key *SK* and associate it with a set of attributes *S'* such that $S' \subseteq S$.

One Way Hashing Functions

A cryptographic hash function is a one way function that maps a large input data to a small fixed size digest. The digest can be used for digital signature, message authentication, and many other security application [25]. Example of hash functions is the SHA family (SHA-1, SHA-256, SHA-512) [24] and lately the newest member of the SHA family SHA-3 [14]. The characteristics of a good hash function are

- Easy to compute the digest from the message.

- Impossible to generate the original message from its hash.
- Any small change in the message must change the hashed value.
- It is very hard to find two messages that have the same hashed value.

The hash functions can be used to prove data correctness and integrity by creating a digest for the data before storing it in the cloud and keep the digest in a safe place. Later when the user wants to prove data integrity she can calculate the digest for the stored data and compare it with the old digest that was stored in a safe place, the two digests are equal if and only if there were no changes in the original data.

1.2.3 No-SQL Databases

No-SQL is a no structure database that was developed to handle the huge, unstructured data generated through the web that requires fast access and does not have a clean and predefined structure [10]. No-SQL systems removed all the constraints and complex operations provided by Relational Database Management Systems (RDBMS), so No-SQL does not support complex operations and joins and does not require a predefined structure for the data to be stored. This gives the NO-SQL the advantage of being fast and horizontally scalable.

No-SQL provides a suitable infrastructure for a medical application, since there is no standard or limit to how much information can be stored about a patient. Also it can be looked at as a key-value pairs since there are no relations between medical data of a person and another from providers point of view.

1.2.4 Electronic Health Records

“The electronic health record (EHR) is an evolving concept defined as a longitudinal collection of electronic health information about individual patients and populations. Primarily, it will be a mechanism for integrating health care information currently collected in both paper and electronic medical records (EMR) for the purpose of improving quality of care" [16]. In addition of the role that EHR systems play in health data sharing and simplifying the process of accessing and searching medical records [16] [17], EHR systems also help:

- Improving quality and convenience of patient care.
- Increasing patient participation in their care.
- Improving accuracy of diagnoses and health outcomes.
- Improving care coordination.
- Increasing practice efficiencies and cost savings.
- Improving clinical and health services research and clinical education.
- Eliminate the physical storage requirements.
- Providing the potential to deliver a longitudinal record that can cover a long track of medications and history for patients and provide comprehensive data across populations.

The ease of accessing and sharing medical data between providers through EHR systems made it also easier for unauthorized users to access private data that was unavailable or very hard for them to access before EHR systems (internal threats). The amount of private and confidential data that is managed by EHR systems made them an active target for attackers and hackers to gain access for sharing, selling, or tampering medical data (external threats). Therefore, EHR systems in addition to traditional needs to preserve patients' privacy, confidentiality, and security, must be able to protect the data and defend against internal and external threats.

The Australian and the U.S. Models

Australia and the U.S. follow different architectures toward health information sharing. [15]. In the Australian model, as illustrated in Figure 1.1, healthcare providers filter

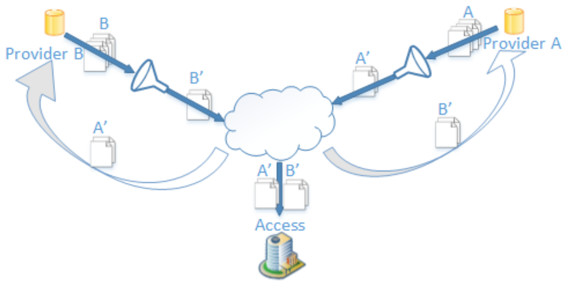


Figure 1.1: Australian Model “Pull-Model”

health data before pushing it into a centralized data center. Thus, a summery of the critical health data is formed and moved from a local health data repository into a centralized system. In this model, providers have control over which information goes into the summary report that will be shared with other healthcare providers. Therefore, during emergencies, only the collection of summery reports will be accessed by providers that contains only meaningful data. The Australian model focuses more on preserving the privacy of patients by making sure that only the needed data is pushed and shared with other healthcare providers, however, the summery report may lack needed information for normal daily usage of the EHR system such as management and financial data.

On the other hand, in the U.S. model, as illustrated in Figure 1.2, local health data held by providers is replicated into a centralized data repository after obtaining consents from patients to be shared with other providers. Thus, during emergencies, healthcare providers can access the complete health data stored in the centralized repository. Though the U.S. model focuses on making sure that all patient data is available for other providers for normal daily usage of the EHR system, it raises privacy and confidentiality concerns specially with emergency access.

Our proposed EHR model will combine the Australian and the U.S. models to provide an efficient privacy-preserving system for emergency care purposes while providing full functionality of an EHR system.

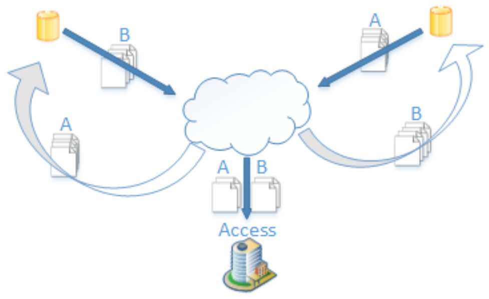


Figure 1.2: US Model “Push-Model”

1.3 Hypothesis

Using a single design to achieve a fully secured system that preserves privacy and confidentiality with the ability to provide fast and easy access to emergency data, is almost impossible without compromising some of the security features in EHR or without providing an easy way to override these security measures. Also even though encrypting data while being at rest or transit is essential to protect the data and preserve privacy and confidentiality, encryption is not enough by itself and extra measures should be taken to secure data [11], because most of private computers have been compromised and users’ lack of security knowledge. The first part of our hypothesis is that EHR systems should follow two different architectures the first is for normal daily access and the other should be for emergency access to be able to get the maximum capabilities in both architectures.

Using the cloud without data auditing and a mechanism to prove data integrity and correctness puts the system trust under suspicion and questioning since there is no guarantee that the data was not tampered with. Our second part of the hypothesis is to make use of the fact that we can use multiple cloud providers to distribute the risk and introduce the ability to audit and prove the correctness and integrity of data by making each cloud stores audit data about another provider.

In order to prove our hypothesis, a prototype system will be built for an EHR system that uses CP-ABE to encrypt and secure medical data over the cloud with fine-grained access control, and an emergency access system that provides the ability to construct a report of critical medical data for emergency access. The data will be split into categories and each category will be stored into different cloud provider also each cloud provider will store a hashed value for data that is stored in one of the other cloud providers. The system then will give users the ability to test integrity and correctness of their data by comparing the calculated hashed value for their stored data with the old hashed value for the same data, that was stored in a different cloud (audit cloud).

1.4 Related Work

1.4.1 EHR system and emergency Access

Many EHR systems have been introduced using the cloud as a data storage service. The Attribute Based Encryption (ABE) family as an encryption scheme to secure data on the cloud and provide fine-grained access control to the users, such as [18] [4] [3] [22]. However, those applications focused on normal daily usage of EHR systems without paying enough attention to emergency access and the impact of emergency access over privacy and confidentiality of the data while being accessed. In the case of [18], the system will only generate a temporary key in emergency cases that grants the user a temporary full access over the patient data without prior authentication. Also, none of the systems introduced the need for separation between emergency data and complete data records for patients during emergency access.

On the other hand, [18] tried to introduce a multi-cloud EHR system architecture, but the introduced system was, in fact, designed over a single cloud provider. The introduced architecture only offered interfacing capabilities for other EHR systems to be able to access their medical data, but the system itself is not using multiple clouds as storage locations for their data.

1.4.2 Data Auditing

Since the cloud providers are not the real owners of the data and they are not authorized to edit, view, or delete the data, many systems were introduced to prove the ownership of the data such as [7], or to verify that the data was not tampered with or deleted by proving its integrity, such as [31]. However, those systems rely on third party auditors to store testing data and keys and to do the verification for them. Using third party auditors may force the users to reveal some private data to the auditors to be able to do their job, which violates the privacy and confidentiality requirements for the system. On the other hand, systems such as [30] were designed to preserve data privacy by making the third party auditors do their job without the need to access any confidential data. Meanwhile, the fact that this system depends on a third party auditor, which is assumed to be trusted to store the auditing data and keys and to do auditing as required from them, is a weakness in the system.

1.5 Road Map

The remainder of this report discusses the implementation and analysis of the construction of emergency report and how to audit the cloud. Section 2.1 and Section 2.2 describe the detailed design and implementation of the system, respectively. Section 2.3 describes the test results of how efficient the emergency report compared with issuing a temporary key to access data. Section 4.1 and section 4.2 describes the current status of the prototype and some ideas for future work, respectively. While conclusion was discussed in section 4.3.

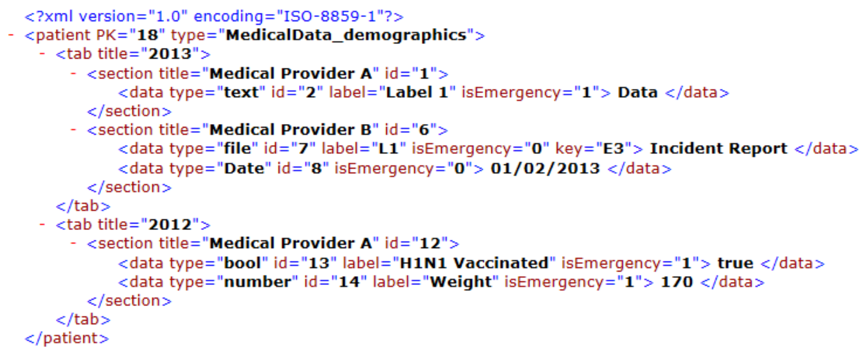
Chapter 2

Design and Implementation

2.1 Design

The proposed framework consists of three entities: a cloud provider (CP), a private secured webserver (PSW), and healthcare providers' private machines (HPM). Patient data is divided into three main categories: medical data, personal data, and insurance data. Each category has many sub-categories for achieving fine-grained access control. Data are stored in a structured XML format (as shown in Figure 2.1). This XML format makes the proposed framework simple to integrate with current EHR systems. Any current system needs only to follow the formatted XML structure to be able to read from or write to our proposed system, which makes it platform independent. Medical records are encrypted using CP-ABE by the patients or healthcare providers before storing them in the cloud. Patients access and manage their medical data via a web application, while healthcare providers can either use the web application to view medical data and emergency reports, or use a desktop version for full capabilities over accessible medical data.

The PSW is initialized by running the setup function of CP-ABE. The function creates the Master Key (MK) and the Public Key (PK). The MK will be saved privately in the PSW without revealing it to other users. On the other hand, the PK will be available for users to be able to encrypt and decrypt data. When healthcare providers register in the system, a desktop application will be installed into their private machines, and the machine will be added to the list of authorized machines to be able to communicate directly with the cloud.



```
  <?xml version="1.0" encoding="ISO-8859-1"?>
- <patient PK="18" type="MedicalData_demographics">
  - <tab title="2013">
    - <section title="Medical Provider A" id="1">
        <data type="text" id="2" label="Label 1" isEmergency="1"> Data </data>
      </section>
    - <section title="Medical Provider B" id="6">
        <data type="file" id="7" label="L1" isEmergency="0" key="E3"> Incident Report </data>
        <data type="Date" id="8" isEmergency="0"> 01/02/2013 </data>
      </section>
    </tab>
  - <tab title="2012">
    - <section title="Medical Provider A" id="12">
        <data type="bool" id="13" label="H1N1 Vaccinated" isEmergency="1"> true </data>
        <data type="number" id="14" label="Weight" isEmergency="1"> 170 </data>
      </section>
    </tab>
  </patient>
```

Figure 2.1: XML format for Data before encryption

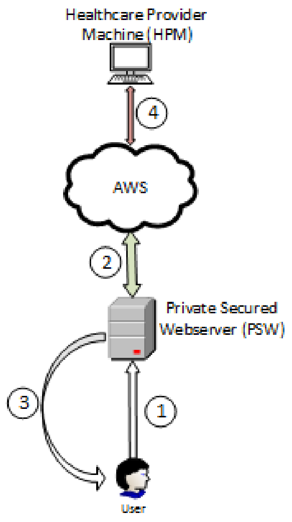

Healthcare Provider Machine (HPM)
4
AWS
2
Private Secured Webserver (PSW)
3
1
User

Figure 2.2: System Structure

The PSW then creates private keys for each registered healthcare provider with identifiable attributes. Also, a unique identification value is given to all users to be able to access the web version of the application, the identification values are also associated with the private keys as an attribute. Patients can request access to the system only through the PSW sever. Private keys and identification values will be created for patients to access the system and decrypt their private data. Patients control who can access their medical data by editing access policies. This can be done by requesting an update process in which PSW will re-encrypt the data with the new access policy, then store it in the cloud.

Figure 2.2 shows the main parts of the system and a high-level overview of the data interactions between each part:

1. A user connects to PSW and provides required authentication data to access the system, and request access to medical data.
2. PSW downloads the user's encrypted data from the cloud using the user's identification value,
3. PSW will decrypt the data using the patient's key and send the decrypted data over secured channels to the users to view it in a web browser.
4. Healthcare providers' machines can also access the cloud directly without accessing PSW for better response time. They load the encrypted data from the cloud by providing the patient identification value, then their local machine can decrypt it if their private keys satisfy the access policies associated with the encrypted data. The providers can also edit the data and access policies, re-encrypt the modified data, then send it back to the cloud.

2.1.1 Emergency Report

The construction of Emergency Report (ER) in our framework follows the push architecture in the Australian model described in subsection 1.2.4. The ER is generated by healthcare providers by marking the required medical data as emergency data (as shown in Figure 2.1) where some nodes have the attribute *isEmergency="1"*. The nodes that are marked as emergency will be copied to another XML file that forms the ER. After finishing selecting all the required nodes for the ER, the system will generate an XML file and store it in the cloud for the ER report. The report will be encrypted in the cloud using Symmetric-key schemes, rather than using CP-ABE for fast access, and The symmetric-key will be stored in the PWS server and in medical providers' private machines.

Healthcare providers can access the ER through their desktop application by only providing the patient identification value, providing that it is an emergency access. Also, providers can access the ER through the web interface, but in this case, they have first to provide a valid authentication for accessing the web before being able to load patient data using patient identification.

Our system provides a hierarchical access policy since data is divided into three main categories: medical data, personal data, and insurance data; Each category is divided into subcategories. This hierarchy gives us the ability to define access policy at each level. Taking medical data as an example, users can give access to all medical data by constructing a general access policy; Then for each sub-category, users also can define another access policy. The system will combine those two access policies into one by *OR* or *AND*, depending on user needs. The final access policy will be {(GeneralAccess)(OR—AND)(SpecificAccess)}, which provides fine-grained access control.

2.1.2 Data Auditing

The system will be built over multiple cloud providers. The data in the system will be divided between those providers, such that the system only knows what data is being stored in each cloud. This architecture distributes the risk among multiple data providers for normal use. The system can make use of this architecture to provide data auditing without the need of a third party user to do the work and without the need for any extra work from the user. This can be done by simply hashing the encrypted data before saving it in cloud A and save the digest in cloud B (as illustrated in Figure 2.3). To provide a mechanism to prove data integrity, the system provides users with an interface to be able to either test one category or all the data in one request. When the user asks for a test to prove integrity, the system will load the encrypted data from cloud A, hash it, then load the old hash value that was stored in cloud B and compare them. This way, if the two hashes are identical, then there was no modification in the data; Otherwise there was something

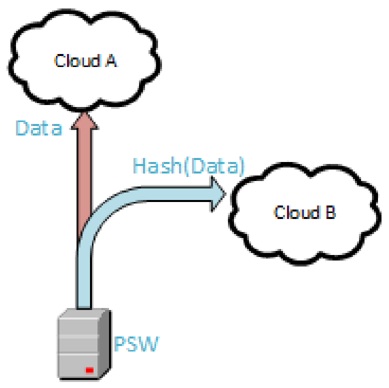


Figure 2.3: Data Auditing

wrong.

Figure 2.4 shows a complete design for the system built over the three major cloud providers (Google, Azure, and Amazon Web Service AWS). Also the figure shows data transfer between each model and the state of the data while being transferred.

2.2 Implementation

A prototype of the system was built on Amazon Web Service (AWS) and Microsoft Azure. The PWS server was built on Amazon Elastic Compute Cloud (EC2). DynamoDB, a NoSQL database, was used as a storage center to store the encrypted medical data in AWS, and NoSQL table service on Azure was used to store the encrypted personal data. Meanwhile, AWS was used as an auditing server for Azure, and Azure was used to audit AWS. The system was built using PHP5 over an Apache2 Server. The EC2 server was configured with PHP5, Apache2, and CP-ABE setup. AWS SDK PHP2 was used to communicate between EC2 and DyanmoDB, and Windows Azure SDK for PHP was used to communicate between EC2 and table service on Azure. For complete installation instructions, please see appendix A.

The prototype was built to demonstrate the functionality and usability of the system by providing a simple web interface for both patients and medical provider. First, the system

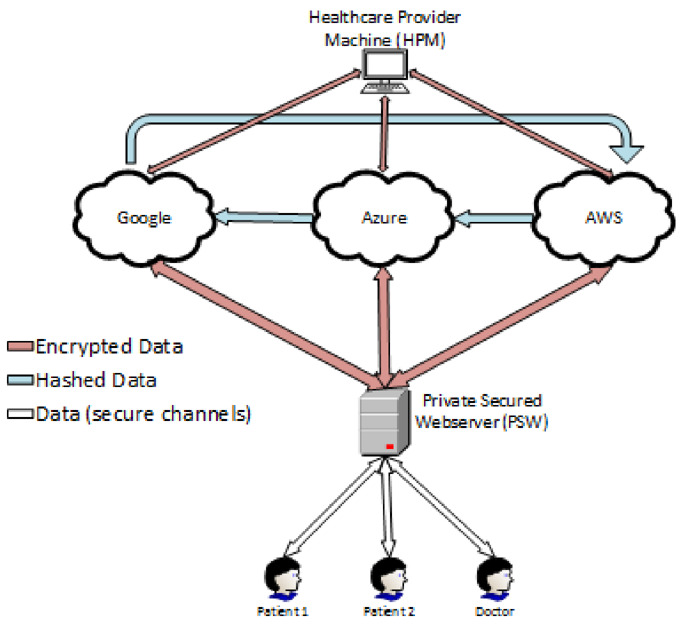


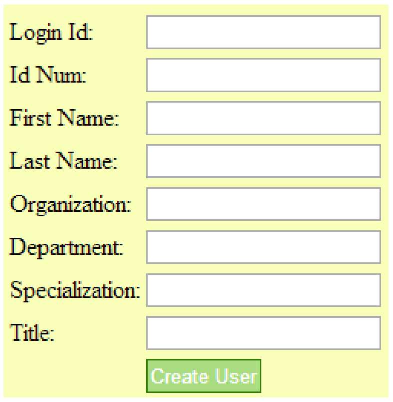
Figure 2.4: Complete System Design

provides an interface to create users by specifying the attributes that identify each user (see Figure 2.5). Then the users can start accessing the system by providing their identification value in the log-in page. A screen will appear for users where data can be viewed, edited, or added (see Figure 2.6). Figure 2.6 explains the different parts of the main screen which consists of:

1. Data Menu: Can be used to navigate among different data types, and to navigate among each data type's subcategories. When a user selects a subcategory, the client will communicate with the server to load the data using the user key. The data will appear if the user has permission; otherwise, a permission denial message will appear.

2. Action menu: Used to manage actions that can be done over data:

    - New Tab: The data inside subcategories is stored inside tabs so users must first create a new tab to add data into it. Tabs are only used to organize data inside each subcategory.



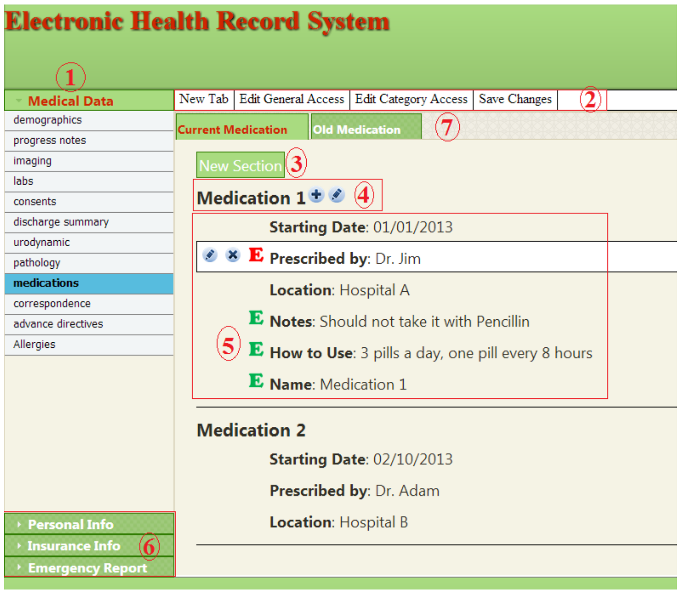
Login Id:	
Id Num:	
First Name:	
Last Name:	
Organization:	
Department:	
Specialization:	
Title:	
	Create User

Figure 2.5: Create New User

- Edit General Access: For achieving fine-grained access control and to make it easier for the user, access policy was divided into two main categories (General Access and Category Access). General access is used to add shared policy over the main category. In our example where the user has selected medical data (as in Figure 2.6), the general access policy will be applied to all the subcategories inside that category (see Figure 2.7 for an example about constructing access policy.)

- Edit Category Access: Used to apply a specific access policy over one of the subcategories (see Figure 2.8). In the first line in the category access, there is an extra join operation, which will be used to join this category access policy with the general policy. Therefore, the user can join the general policy with the category policy with either AND or OR operations according to the user needs.

- Save Changes: All the changes that the user did to the data or access policy will not be reflected to the cloud unless the user hit (save changes). Even if the user moved between categories or subcategories, data changes will be saved in the session until hitting (save changes). When the user clicks on (save changes), the system will search for all the changes that has been made to the data, encrypt the data, calculate the hash digest, then save the data in the data cloud and the digest in the audit cloud.



Medication 1

Starting Date: 01/01/2013

E Prescribed by: Dr. Jim

Location: Hospital A

E Notes: Should not take it with Pencillin

E How to Use: 3 pills a day, one pill every 8 hours

E Name: Medication 1

Medication 2

Starting Date: 02/10/2013

Prescribed by: Dr. Adam

Location: Hospital B

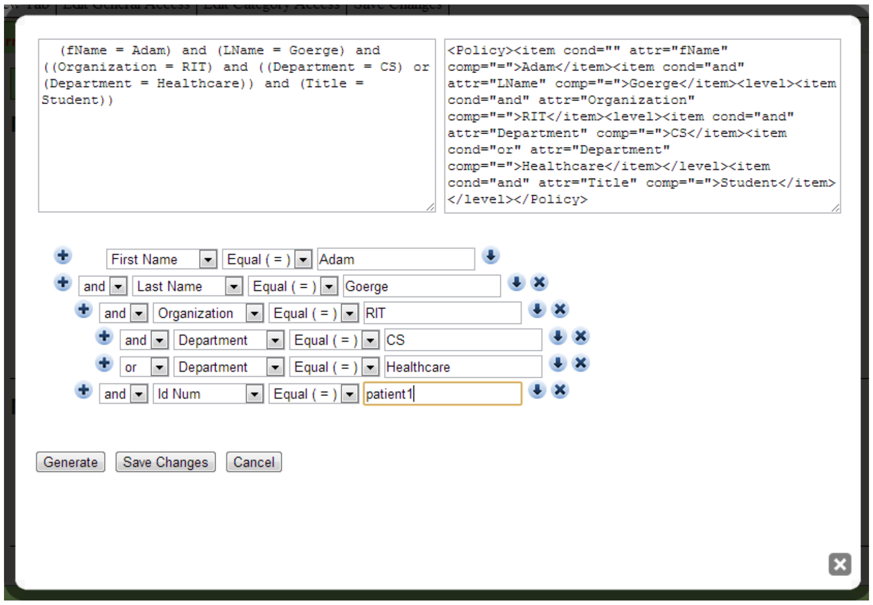
Figure 2.6: EHR User Interface

3. New Section: Each tab contains multiple sections to organize data. The user can add sections to the tab by using the (New Section) button and by providing a title for that section. In Figure 2.6 (Medication 1) represents a section in (Current Medication) tab.

4. The user can add data to sections by clicking on the blue plus sign near the section title (see Figure 2.6 mark 4). The user can provide a label and text for each line of data.

5. In Figure 2.6, the data that have green **E**s preceding them are marked as emergency data. The others are not included in the emergency report. The user can mark the data as emergency by clicking on the red **E** preceding the data, or clicking the green **E** to remove the data from the emergency report.

After uploading all the data to the cloud, the user can start building the emergency report by selecting the needed data,then the system will construct an emergency report (in XML format). When saving the report, the system will encrypted it using AES with a private key

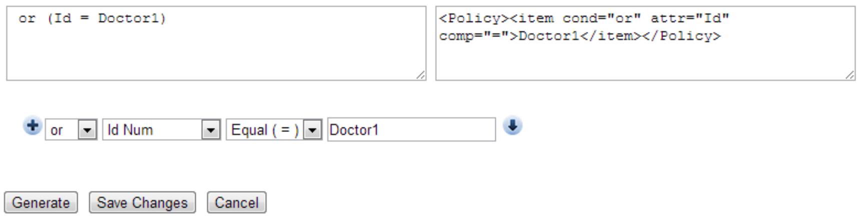


```
 (fName = Adam) and (LName = Goerge) and
((Organization = RIT) and ((Department = CS) or
(Department = Healthcare)) and (Title =
Student))
```

```
<Policy><item cond="" attr="fName"
comp="=">Adam</item><item cond="and"
attr="LName" comp="=">Goerge</item><level><item
cond="and" attr="Organization"
comp="=">RIT</item><level><item cond="and"
attr="Department" comp="=">CS</item><item
cond="or" attr="Department"
comp="=">Healthcare</item></level><item
cond="and" attr="Title" comp="=">Student</item>
</level></Policy>
```

	First Name	Equal (=)	Adam
and	Last Name	Equal (=)	Goerge
and	Organization	Equal (=)	RIT
and	Department	Equal (=)	CS
or	Department	Equal (=)	Healthcare
and	Id Num	Equal (=)	patient1

Figure 2.7: General Access Policy



```
or (Id = Doctor1)
```

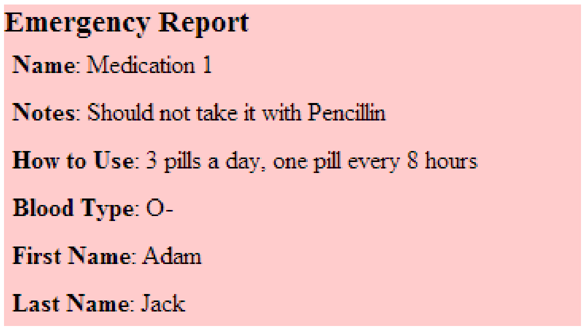
```
<Policy><item cond="or" attr="Id"
comp="=">Doctor1</item></Policy>
```

or	Id Num	Equal (=)	Doctor1

Figure 2.8: Category Access Policy

only available at PWS and providers' registered machines. Firgue 2.9 shows an emergency report that was constructed from both medical data and personal data for a random patient. In the current prototype, the report is stored in a NoSQL dynamoDB table in AWS, but for availability and access speed, the report can be distributed to more than one cloud provider if needed.

In the NoSQL databases, the system uses two keys to navigate between different users and to uniquely identify subcategories in a single record for a patient. The main key (primary key) is the user unique identification value, which is used to uniquely identify patient records. This key can also be used to scan and pull all data records for a patient using scan



Emergency Report

Name: Medication 1

Notes: Should not take it with Pencillin

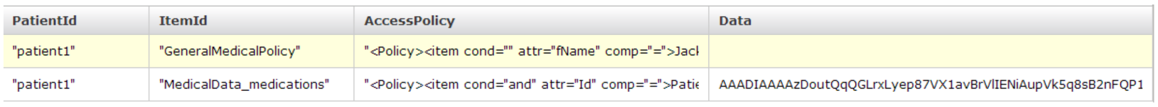
How to Use: 3 pills a day, one pill every 8 hours

Blood Type: O-

First Name: Adam

Last Name: Jack

Figure 2.9: Emergency Report



PatientId	ItemId	AccessPolicy	Data
"patient1"	"GeneralMedicalPolicy"	"<Policy><item cond="" attr="fName" comp="=">Jacl	
"patient1"	"MedicalData_medications"	"<Policy><item cond="and" attr="Id" comp="=">Patie	AAADIAAAAzDoutQqQGLrxLyep87VX1avBrVlIENiAupVk5q8sB2nFQP1

Figure 2.10: Data in The Cloud

methods. The second key (range key) is the subcategory title (see Figure 2.10), which is used to uniquely identify each subcategory for a single patient. The range key is used to pull a single subcategory when a user navigate among subcategories for better response time.

2.3 Testing

In order to test the idea of building an emergency report in an EHR system and how efficient it will be compared to just issuing an emergency key and decrypting all medical data, 10 patients, 2 doctors, and 2 nurses accounts were created. Then some dummy medical data were added for each patient. The EHR system was tested with CP-ABE access management by creating different access polices for each patient, and was tested for each doctor to access patients Data. Doctors were able to access only the authorized data, and patients were able to create fine-grained access policy over subcategories in medical data and personal data. These scenarios verified that the system works as a basic EHR system with fine-grained access control over CP-ABE.

In order to test the emergency report functionality, different kinds of reports were created for patients. One of the patients had all of her medical and personal data marked as emergency, while other patients had different number of their attributes marked as emergency data. Then these three scenarios were tested:

- Asking unauthorized doctors to access emergency data. The doctors were only able to access the emergency reports and were only able to see the data that was marked as emergency. There was no need to access the total medical records for patients, and no need for revealing any unnecessary data.
- The doctors was able to update data that was marked as emergency, and changes were reflected correctly to the emergency report after saving the changes.
- Removing some marked data from emergency report, was also reflected correctly to the report after saving the changes.

In order for us to compare between the two systems, we created two scenarios. In the first scenario, every single attribute in the patient record was marked as emergency data, This proves that our system provides the same access to all data as the traditional systems do, but with extra overhead when updating data, since the data is replicated in two places. Then the system was tested for access speed. First we used the emergency key to access emergency data (as in the traditional system using temporary key) and got an average time of 0.44 seconds to decrypt all the data. On the other hand, the system was able to access the same data in 0.031 seconds when reading emergency data in the emergency report that was encrypted by AES.

The other scenario resembles the real scenario where the needed medical data for treating patients is a small part of the complete medical record of the patient. In this scenario half of the attributes in the medical record for the patient were marked as emergency data. The first part of the test, when accessing the system using the emergency temporary CP-ABE key, the decryption process took an average time of 0.43 seconds. This time was almost the same as the first scenario, which is expected since temporary key method in both scenarios is accessing the same size of data. On the other hand, accessing the emergency report data took only 0.028 seconds, which is also expected since the system is accessing half the data that was used in the first scenario.

Audit functionality was tested by designing the system to work under two modes. In the first mode, which is the normal mode, the system will audit the data by first encrypting the medical record, then it will calculate the hash digest for the encrypted data. after that the system will save the encrypted data in the data cloud, and the digest in the audit cloud. However, the other mode saves only the encrypted data without calculating its hash digest. The second mode resembles modifying the data by a third party such as the cloud provider or an attacker without going through the system. In order to be able to verify the data integrity, an interface was created where a patient ID can be provided and request a verifying process. The system will then load the encrypted data for that patient from the data cloud, calculate the hash digests each subcategory, and then compare them with the old digests that were stored in the audit cloud. Finally, the system will generate a report about which data was tampered with and which was not.

Chapter 3

Analysis

3.1 Emergency Report

Cryptography is becoming less and less important in the world of defending data and systems, as Shamir (one of the fathers of public-key cryptography) stated at the 2013 RSA conference [11]. Therefore, the proposed EHR system provides another level of defense against attacks, especially for EHR systems in emergency access were data is easily exposed. The system protects privacy and confidentiality by giving the ability to limit the amount of data that is exposed to users in emergency access situations. These limitations prevent attackers from gaining access to information that may harm both users and providers, yet the system was able to do its job by listing all the critical data needed for emergency.

On the other hand, speed is a very important factor when developing a national scale EHR system, especially for emergency access to be able to deliver on-time emergency treatment. The huge size of medical data and the large number of patients adds a huge overhead on the EHR core. This overhead makes the system slow and unusable, especially for emergency access. Therefore, other measurements and core designs must be taken to improve the system. The tests that were done in section 2.3 prove that the response time (0.031 sec) for emergency reports was much faster than the response time (0.44 sec) for using temporary keys in CP-ABE. This speed makes our system more suitable for a national scale EHR system. The speed in our system results from:

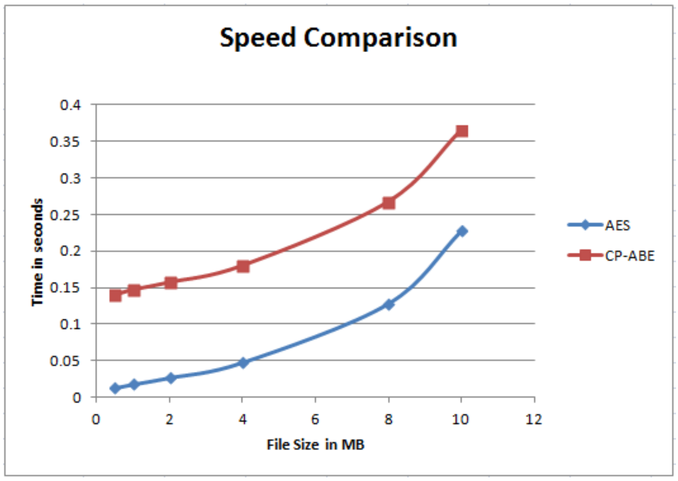


Figure 3.1: CP-ABE vs. AES

- Using symmetrical encryption schemes in securing emergency report, which is the simplest and fastest form of encryption. This compares to traditional EHR systems that use CP-ABE to secure data in the cloud, which is very much slower than symmetrical encryption. This was proven in the testing done section 2.3, in [4]. Figure 3.1 shows the difference in speed between AES and CP-ABE when used to encrypt different sizes of files.

- The size of the data needed to be encrypted and decrypted plays a major role in the speed of the operations. Since the size of the data blocks encrypted in the schemes is constant, increasing the size of the data generates more and more blocks, which means slower operations. Therefore, by marking only the critical data and generating smaller reports, emergency access will be faster. The effect of the file size on speed can also be seen in Figure 3.1.

- The number of operations needed to access the data. In the case of EHR systems without emergency report, every subgroup that contains critical data must be decrypted. In our system, medical data has twelve subgroups, which means we need twelve decryption operations to be able to access all the needed data. CP-ABE by itself is slower than AES, and the need to apply it more than once slows the system

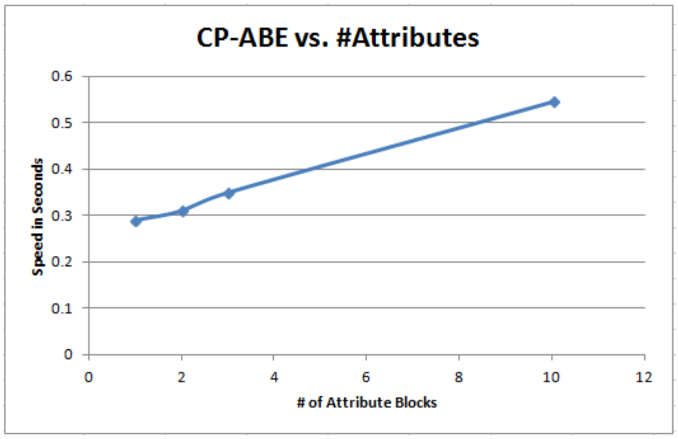


Figure 3.2: Relation between Speed and Access Policy Complexity

very much. Comparing this number of operations with the need of only one operation to access emergency report, in addition of using AES which is faster, makes our system much faster in emergency situations.

- Access policy complexity also plays major role in the speed of CP-ABE. Figure 3.2 measures the encryption speed of CP-ABE algorithm when encrypting a 10 MB file using different access policies. As we can see, the speed of the encryption decreases, while the size of the access policy increases.

The separation between EHR and emergency access is not only good for emergency access, but also for EHR systems. Before this separation, EHR systems were forced to provide a mechanism that enables users to access data without permission in case of emergency. This mechanism introduced a burden and some security holes in the system. Therefore, by removing this burden, EHR systems can now focus on creating a more secure system that focuses on protecting privacy and confidentiality for patient data by forcing access policies over data as it supposed to be without exceptions.

3.2 Key Management

Key management in the system handles two types of keys. The first one is related to the CP-ABE keys and its daily use in the EHR system. While, the other one is related to AES encryption scheme and its use in emergency access. Also each type can be analyzed from both patients and providers point of view.

3.2.1 CP-ABE

There are three types of keys in CP-ABE master (key, public key, and private key):

- Master Key: A private key that is used to generate users private keys. This key is only stored in the PSW which acts as the key authority in the system. This key is kept private and it will not be shared with any user, provider, or third party that may have access to the system.
- Public Key: A public key that is needed to encrypt or decrypt data in CP-ABE systems. This key will be shared with all users, providers, and systems that may have access to the system.
- Private Key: A unique key that will be generated for each entity, when it join the system. This key uniquely describes each entity with the associated attributes. The users will use their keys to gain access to data that was authorized for them. This key should be accessible only by its owner.

3.2.2 Emergency Report key

The emergency report section uses symmetrical key encryption, where only one key is required for encrypting and decrypting data. The key will be stored only in two places. The first place is the PSW server, in order to be able to access the emergency report through the web interface. The second place is the registered private machines for each provider who is authorized to gain emergency access.

3.2.3 Provider Keys

Providers can use the system either through their private desktop applications or through the web interface. When using the private desktop application, only regular authentication is required, such as active directory or user name and password through the provider's database, to access the system. After authentication, the provider can start accessing medical data using the system, since the required keys (public key, private key, and Emergency key) are stored securely in the application.

On the other hand, there are two modes for the providers when they use the web interface. If the provider wants to access only emergency access through the web interface, a user-name and password can be used to authenticate providers. After that the provider can access patients emergency data since the required key is already stored in the server. However, providers need to upload their private CP-ABE keys to PSW, as an extra step, in order to gain access to complete medical records. Therefore, the web interface provides fast and easy way to access emergency data for any patient, without the need to download or upload any application or keys, and provides a reasonable mechanism to access complete records.

3.2.4 Patient Key

Patients can only use the web interface to gain access to their data, which is encrypted using CP-ABE. For simplicity and better user experience, the patients will only be authenticated using user-name and password. The system will either load a locally stored key for the patient, or recreate the access key, since both the public and master key are stored in PSW. In order for the patients to gain access to their encrypted data, the system modifies the access policy for each sub-category by attaching the patient's identification key to the access policy. The original access policy looks like (GeneralAccess [OR/AND] Category-Access), which defines the attributes of the users who have access to the data. while, The modified version looks like ((GeneralAccess [OR/AND] CategoryAccess) OR PatientId), with this modification, the patient key ,which only contains the identification key for the patient, can access the encrypted data.

3.2.5 Key Analysis

The key management in our system enhance the user experience. For providers, accessing medical data for daily normal use will be done mostly from their private machines. Those machines already registered and authenticated in our system and store the required keys locally. Therefore, providers are not required to carry and provide their private keys, which minimize the risk of compromising those keys. On the other hand, emergency access must be accessible anytime, anywhere, and as fast as possible. Therefore, medical providers can use the web interface to log-in the system and gain access to emergency data. Since the symmetrical key is already stored in the PSW server, there is no need for uploading or generating any temporary keys. However, if the provider wants to access the complete records through the web interface, the CP-ABE private key for that provider must be uploaded.

In order to make sure that the web interface can be usable for providers to gain access to complete medical records, a comparison between the key size and the number of attributes attached to the key was done. Figure 3.3 shows the relation between the key size and the number of attributes attached to that key. As seen in the figure, the initial size of the key with one attribute is less than 0.5 kb, and adding extra $\sim$250 bytes to the key size for each addition attribute. This comparison shows that the key is very small and can be uploaded easily to the server.

One may argue that since the key with one attribute (patient key) is very small, then why cannot we also ask the patients to upload their keys. The lack of security awareness, usability,the easiness of losing such a small file or share it with others, and the diversity of patients background make it safer to stick with id and password authorization instead of asking the users to upload their keys.

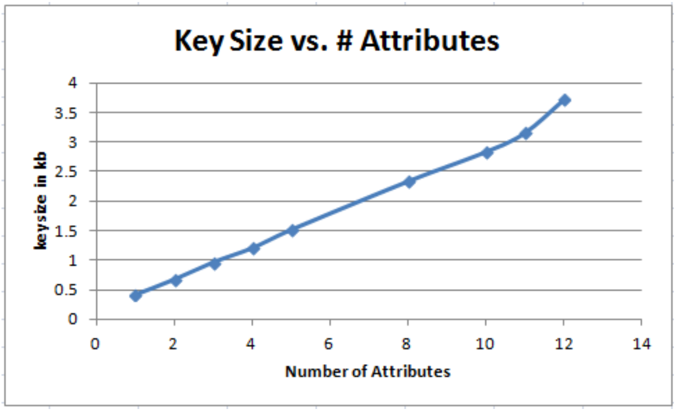


Figure 3.3: Relation between Key Size and Number of Attributes

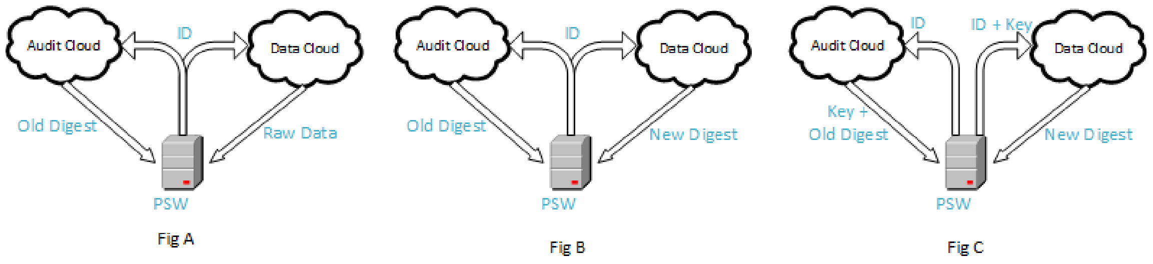


Figure 3.4: Different Architectures for Cloud Auditing

3.3 Data Audit Architectures and Calculations

Data auditing plays a major role in any cloud system, since the data is not under the direct control of the owners anymore. Data auditing in the cloud was introduced before (such as [30]), but this system requires a trusted third party to handle the auditing work. Also, the system requires some extra work from the users themselves. Our auditing system makes use of the multi-cloud infrastructure to build a self-auditing system without the need of any third-party auditor. In our system, each cloud provider plays two roles. The first role is a data storage system in which our system stores part of patient data. The second role is an auditor in which each cloud provider stores the auditing data for one of the other clouds. Figure 3.4 illustrates three design architectures for auditing in the cloud, as follows:

- Figure A: Illustrates the architecture in which the PSW server will send the data ID to both clouds requesting data. After loading the old digest from the audit cloud and loading the raw data from the data cloud, the system will calculate the digest for the raw data locally, then compare it with the old digest. The problem with this architecture is the need to download the complete raw data from the data cloud to be able to calculate the digest.

- Figure B: Describes the architecture in which the PSW sends the ID to both servers. The PSW will receive the old digest from the audit cloud, while the data cloud will first calculate the digest for the raw data and send it to the PSW instead of the raw data. The digest is much smaller and can be downloaded very fast. Then the PSW will compare between the two digests to make sure that there were no changes to the raw data. This architecture can be used with a semi-trusted data cloud. The semi-trusted cloud is trusted to calculate the digest correctly, but not trusted to store the data correctly.
- Figure C: The previous two architectures use hash algorithms, such as *SHA512*, to calculate the digest without any requiring a key from the user. However, The architecture in Figure 3.4-C uses a cryptographic hash function in which the user should provide a key for the hash function in order to get the correct digest. Therefore, PSW will first load the old digest and the key used to calculate that digest from the audit cloud; PSW will then send the ID and key to the data cloud. The data cloud will use the key to calculate the new digest and send it to PSW. PSW will compare between the two digests to verify data integrity.

The architecture in Figure3.4-C can be used with untrusted cloud providers. An untrusted cloud provider can calculate the digest of the data when it was uploaded. When users want to validate their data, they will receive the old digests, which do not reflect the current state of the data. Therefore, using Figure C architecture, the cloud providers will be forced to calculate the digests when users ask for a validation process, not when uploading the data, since this cannot be done without the keys that are stored in the audit cloud.

	Load Digest	Load Data	Calc Digest	Total Time
1	0.56	1.78	0.007	2.34
2	0.12	1.22	0.007	1.35
3	0.12	1.13	0.007	1.26
4	0.10	1.74	0.007	1.85
5	0.11	1.59	0.007	1.70
6	0.10	1.60	0.007	1.71
7	0.10	1.55	0.007	1.65
8	0.14	1.71	0.007	1.86
9	0.12	1.52	0.007	1.64
10	0.11	1.68	0.007	1.80

Table 3.1: Operations Length in Figure 3.4-A

Table 3.4 lists an average time comparison between the three architectures in Figure 3.4. Each operation were done 10 times, Figures [3.1,3.2,3.3]. All results are in seconds and are done over a (1 MB) file.

- Load Digest: The time needed to load the data from the audit cloud. In this case, Azure was the audit cloud.
- Load Data: The time needed to download either the raw data (in Figure A case), or to load the calculated digest for the raw data (in Figure B and C cases) from the data cloud. In this case, AWS was the data cloud.
- Calc Digest: The time needed to calculate the digest for the raw data.
- Total Time: The total time needed by the server from receiving a request from the client, until sending the results into response.

The difference between design A and design B is the place where the new digest is being calculated. Design A downloads the complete record then calculate its digest. This design advantage lies in removing the burden of calculating the digest from the cloud or the data server, but the fact that it needs to download the complete record is a big disadvantage.

	Load Digest	Load Data	Calc Digest	Total Time
1	0.46	0.14	0.009	0.60
2	0.09	0.13	0.008	0.22
3	0.11	0.12	0.008	0.23
4	0.09	0.12	0.008	0.21
5	0.10	0.12	0.008	0.22
6	0.09	0.12	0.008	0.21
7	0.09	0.11	0.008	0.20
8	0.10	0.13	0.008	0.23
9	0.09	0.12	0.008	0.21
10	0.10	0.11	0.008	0.20

Table 3.2: Operations Length in Figure 3.4-B

	Load Digest	Load Data	Calc Digest	Total Time
1	0.23	0.13	0.008	0.36
2	0.10	0.14	0.008	0.23
3	0.10	0.13	0.008	0.23
4	0.10	0.11	0.008	0.21
5	0.10	0.12	0.008	0.22
6	0.13	0.14	0.008	0.26
7	0.10	0.10	0.008	0.20
8	0.09	0.12	0.008	0.20
9	0.13	0.11	0.008	0.24
10	0.09	0.12	0.008	0.21

Table 3.3: Operations Length in Figure 3.4-C

	Load Digest	Load Data	Calc Digest	Total Time
Figure A	0.16	1.55	0.007	1.72
Figure B	0.13	0.12	0.008	0.26
Figure C	0.12	0.12	0.008	0.24

Table 3.4: Average Times Comparison between Cloud Auditing Architectures

Therefore, this design is applicable in local network where transferring data is not a bottleneck. Whereas, design B calculate the digest for the complete record in the cloud and download the digest only. Design B is better than design A in our case, since data transfer is a bottleneck in our case and we can depend on the cloud powerful resources to calculate the digest of the data, but the owners will lose their full control over the calculation operations. Therefore, design C comes to make sure that the cloud provider is doing their job as it should be without any cheating. Also design C can be improved to protect our system from any cooperation between the audit cloud and the data cloud by either storing the keys locally or in a third cloud provider, since already our system was designed to distribute data between at least three cloud providers.

Chapter 4

Conclusions

4.1 Current Status

- The application is only a prototype which was built to show that our hypothesis does have an add value and applicable in providing the current EHR systems tasks in addition of preserving patient's privacy and confidentiality.
- The web application is used by both doctors and patients without any distinguishes between them, we need to separate between both interfaces and the capabilities provided for each type.
- The desktop application for medical providers was not built since we were able to provide the jobs that need to be done through the web interface.
- Only medical data and personal data were implemented in the system we need to include other parts such as insurance, medical history, family history...etc.
- The system was only built over Azure and AWS since those two were enough to prove the audit capabilities and enough to prove the main concept behind the hypothesis. Therefore, we need later to include Google and other cloud providers for different kinds of data.

4.2 Future Work

4.2.1 Replication

The idea of moving data to the cloud and not have it locally, may be rejected by some medical institutions, especially large ones. In order to solve such an issue, the system can provide a replication mechanism, in which each medical provider can mark its clients. The system will start replicating each provider's list to their local database and keeps updating the list with any new changes. The system also should give users the ability to track the list of providers that have local copy for their records, and give the users also the ability to delete or prevent some providers from storing their data locally.

4.2.2 Identification List

Identification list is a list that contains pairs of words, such as (Adam:Patient_First_Name). The first word is an identification word such as patient name, disease name, doctor name ...etc. The second word is a pseudonymous word that hides the identification meaning of the first word without removing the meaning of the data. This list can be used to hide identification information in emergency reports or later in any other report that the system can generate. This way we can make sure that the reports that are easily accessed can only be accessed through the system. Otherwise, the data has no identification information, which is an extra measure to preserve privacy and confidentiality.

4.2.3 Emergency Declaration System

The system needs an advanced infrastructure to be able to identify emergency situations and users who can access emergency data. The system should prevent using emergency data in normal daily situations to prevent data miss-using. Also the system should be able to identify emergency situations without any overhead.

4.2.4 Alert and Notification System

The system needs an alert and notification system to give both providers and patients the ability to track their data. Alerts can help users to prevent any miss-use of their data. Also the alert system can be used to identify suspicious activities. The user should be notified when a provider requests a local copy of their data or accessed their emergency report. The system should be able to detect suspicious actions from users and notify administrators to take action as soon as possible to prevent any extra damage.

4.2.5 Research Reports

The idea of creating a summary report of needed data and hide identification data, without altering the meaning of the information, can be used to provide research reports. The system can provide the ability to generate reports like the emergency report, by giving users the ability to mark the data that can be used. The system will then hide the identifying information, then push the reports to research facilities. Those reports can be used in research or generating reports and charts without the need to reveal any private or secured data.

4.3 Conclusion

Creating an emergency report from patient data to be accessed in emergency situation, is an additional measure to preserve patient's privacy and confidentiality. The system mostly depends on the cooperation between the patient and medical provider, and between medical providers themselves to achieve its main goal. If the providers did not take time to mark the emergency data and just marked everything then the system becomes useless, it would become a burden on the EHR system and becomes a major security hole in the system.

Bibliography

[1] Chronology of data breaches. `http://www.privacyrights.org`, feb 2012. Accessed: 02/23/2012.

[2] Divyakant Agrawal, Sudipto Das, and Amr El Abbadi. Big data and cloud computing: current state and future opportunities. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 530–533, New York, NY, USA, 2011. ACM.

[3] Joseph A. Akinyele, Matthew W. Pagano, Matthew D. Green, Christoph U. Lehmann, Zachary N.J. Peterson, and Aviel D. Rubin. Securing electronic medical records using attribute-based encryption on mobile devices. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, SPSM '11, pages 75–86, New York, NY, USA, 2011. ACM.

[4] S. Alshehri, S.P. Radziszowski, and R.K. Raj. Secure access for healthcare data in the cloud using ciphertext-policy attribute-based encryption. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, pages 143 –146, april 2012.

[5] Amazon.com. Amazon s3 availability event: July 20, 2008. `http://status.aws.amazon.com/s3-20080720.html`, Feb 2013. Accessed: 02/23/2013.

[6] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing, Feb 2009.

[7] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 598–609, New York, NY, USA, 2007. ACM.

[8] Josh Benaloh, Melissa Chase, Eric Horvitz, and Kristin Lauter. Patient controlled encryption: ensuring privacy of electronic medical records. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, pages 103–114, New York, NY, USA, 2009. ACM.

[9] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 321–334, Washington, DC, USA, 2007. IEEE Computer Society.

[10] R. Burtica, E.M. Mocanu, M.I. Andreica, and N. Tapus. Practical application and evaluation of no-sql databases in cloud computing. In *Systems Conference (SysCon), 2012 IEEE International*, pages 1 –6, march 2012.

[11] Dennis Fisher. Rsa conference 2013: Experts say it's time to prepare for a 'post-crypto' world. `http://goo.gl/d6Gmi`, Feb 2013. Accessed: 03/02/2013.

[12] Free Software Foundation. Gmp arithmetic without limitations. `http://gmplib.org/`. Accessed: 03/17/2013.

[13] Nancy Gohring. Google's app engine breaks down. `http://www.cio.com/article/400013/Google_s_App_Engine_Breaks_Down`, Jun 2008. Accessed: 02/23/2013.

[14] Michal Peeters Guido Bertoni, Joan Daemen and Gilles Van Assche. Keccak (sha-3). `http://keccak.noekeon.org/`, Jan 2013. Accessed: 02/24/2013.

[15] D. Tracy Gunter and P. Nicolas Terry. The emergence of national electronic health record architectures in the united states and australia: Models, costs, and questions. *J Med Internet Res*, 7(1):e3, Mar 2005.

[16] D. Tracy Gunter and P. Nicolas Terry. The emergence of national electronic health record architectures in the united states and australia: Models, costs, and questions. *J Med Internet Res*, 7(1):e3, Mar 2005.

[17] HealthIT. Benefits of electronic health records (ehrs). `http://www.healthit.gov/providers-professionals/why-adopt-ehrs`, Dec 2012. Accessed: 02/14/2013.

[18] Jie Huang, M. Sharaf, and Chin-Tser Huang. A hierarchical framework for secure and scalable ehr sharing and access control in multi-cloud. In *Parallel Processing*

Workshops (ICPPW), 2012 41st International Conference on, pages 279 –287, sept. 2012.

[19] Brent Waters (advisory role) John Bethencourt, Amit Sahai (advisory role). Advanced crypto software collection. `http://acsc.cs.utexas.edu/cpabe/`. Accessed: 03/17/2013.

[20] Robert Slocum Keith Tyson. A focus on prevention is the best remedy for medical record breaches. `http://goo.gl/xSNhQ`, fall 2012. Accessed: 02/18/2013.

[21] Ben Lynn. Pbc library the pairing-based cryptography library. `http://crypto.stanford.edu/pbc/`. Accessed: 03/17/2013.

[22] Shivaramakrishnan Narayan, Martin Gagné, and Reihaneh Safavi-Naini. Privacy preserving ehr system using attribute-based infrastructure. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, CCSW '10, pages 47–52, New York, NY, USA, 2010. ACM.

[23] U.S. Department of Health and Human Services. Hipaa - the privacy rule. `http://www.hhs.gov/ocr/privacy/hipaa/administrative/privacyrule/index.html`, Jan 2013. Accessed: 01/14/2013.

[24] National Institute of Standard and Technology NIST. Secure hash standard (shs). `http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf`, Mar 2012. Accessed: 02/24/2013.

[25] National Institute of Standard and Technology NIST. Cryptographic hash algorithm competition. `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`, Feb 2013. Accessed: 02/24/2013.

[26] amednews staff PAMELA LEWIS DOLAN. E-prescribing soars among doctors with ehrs. `http://www.ama-assn.org/amednews/2012/12/17/bisd1218.htm`, Dec 2012. Accessed: 12/19/2012.

[27] Christine Arevalo Rick Kam. A glimpse inside the $234 billion world of medical fraud. `http://www.govhealthit.com/news/glimpse-inside-234-billion-world-medical-id-theft`, feb 2012. Accessed: 02/18/2013.

[28] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

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

[30] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *INFOCOM, 2010 Proceedings IEEE*, pages 1 –9, march 2010.

[31] Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *Proceedings of the 14th European conference on Research in computer security*, ESORICS'09, pages 355–370, Berlin, Heidelberg, 2009. Springer-Verlag.

Appendix A

Installing Manual

This manual will help the user to create the web server with CP-ABE, PHP5, Apache, AWS sdk, and Azure sdk. The installation was done on Ubuntu Server 12.04.1 LTS which is an EC2 server in AWS.

A.1 Preparing the Server

1. we need to install essential packages for the server (make, gcc, g++ compilers):

   ```
   $ sudo apt-get update
   $ sudo apt-get upgrade
   $ sudo apt-get install build-essential
   $ gcc -v
   $ make -v
   ```

2. Installing required libraries for the system:

   - Installing M4 a macro processor language needed for arithmetic operation library (GMP):

     ```
     $ sudo apt-get install m4
     ```

   - Installing OpenSSL library:

     ```
     $ sudo apt-get install openssl
     ```

   - Installing development version of OpenSSL to get access to encryption libraries:

     ```
     $ sudo apt-get libssl-dev
     ```

- Install Glib library which is needed in CP-ABE installation:

  ```
  $ sudo apt-get install libglib2.0-dev
  ```

A.1.1 Installing GMP

"GMP is a free library for arbitrary precision arithmetic, operating on signed integers, rational numbers, and floating point numbers" [12].

1. Download GMP version 5.1.1:

   ```
   $ sudo wget ftp://ftp.gnu.org/gnu/gmp/gmp-5.1.1.t
   ```

2. decompress the library:

   ```
   $ tar xjf gmp-5.1.1.tar.bz2
   ```

3. To install the library, move inside the folder then call

   ```
   $ ./configure --disable-share --enable-static
   $ make
   $ make check
   $ sudo make install
   ```

   Note: You have to disable the shared functions and enable the static ones, otherwise CP-ABE libraries will not work (configure –disable-share –enable-static)

A.1.2 Installing PBC

PBC is a pairing-Based Cryptography library, which is built over GMP library and designed to be the infrastructure for pairing-based cryptosystems [21]. For the system we need both the development version and the user version:

1. Download user version:

```
$ sudo wget
 http://voltar.org/pbcfiles/libpbc0\_0.5.12\_amd64.deb
```

2. Download development version:

```
$ sudo wget
 http://voltar.org/pbcfiles/libpbc-dev\_0.5.12\_amd64.deb
```

3. libgmp3c2 shared library was needed by PBC, to install it:

```
$ sudo apt-get install libgmp3c2
```

4. Install user version:

```
$ sudo dpkg -i libpbc0\_0.5.12\_amd64.deb
```

5. Install development version:

```
$ sudo dpkg -i libpbc-dev\_0.5.12\_amd64.deb
```

A.2 Installing and Testing CP-ABE

CP-ABE is divided into two libraries,*libbswabe* (a library implementing the core crypto operations) and *cpabe* (higher level functions and user interface) [19]. We will first install those library, then we will do a small test to make sure that everything is working fine.

1. Download libbswabe:

```
$ sudo wget
 http://acsc.cs.utexas.edu/cpabe/libbswabe-0.9.tar.gz
```

2. Download cpabe:

   ```
    $ sudo wget
    http://acsc.cs.utexas.edu/cpabe/cpabe-0.11.tar.gz
   ```

3. Installing libbswabe after decompressing the folder and moving inside it:

   ```
    $ sudo tar xzf  libbswabe-0.9.tar.gz
   ```

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

4. Installing cpabe after decompressing the downloaded file and moving inside it:

   ```
   $ sudo tar xzf cpabe-0.11.tar.gz
   $ ./configure
   ```

   Before calling *make*, we need to edit the *MakeFile* that was created from the first command. The installation did not work without it, I think it is something due to the changes in the GMP library (new version). In the *MakeFile* change the following:

   ```
   LDFLAGS = -O3 -Wall \
           -lglib-2.0   \
           -Wl,-rpath /usr/local/lib -lgmp \
           -Wl,-rpath /usr/local/lib -lpbc \
           -lbswabe \
           -lcrypto -lcrypto
   ```

   To:

   ```
   LDFLAGS = -O3 -Wall \
           -lglib-2.0   \
           -Wl,-rpath /usr/local/lib -lgmp \
   ```

```
        -Wl,-rpath /usr/local/lib -lpbc \
        -lbswabe \
        -lcrypto -lcrypto \
        -lgmp

$ make
$ sudo make install
```

A.2.1 Testing CP-ABE installation

1. CP-ABE setup:

```
$ cpabe-setup
$ ls
master_key  pub_key
```

2. Generate user keys:

```
$ cpabe-keygen -o Doc1_priv_key pub_key master_key \
    Job_Doctor CS_department RIT_Organization \
    FirstName_Adam ID_D1
$ cpabe-keygen -o Patient1_priv_key pub_key master_key \
    Job_Student FirstName_Jack LastName_Tom \
    RIT_Organization Id_P1
$ ls
master_key  pub_key  Doc1_priv_key  Patient1_priv_key
```

3. Encrypt a file called medicalData.xml:

```
$ ls
pub_key  medicalData.xml
$ cpabe-enc pub_key medicalData.xml
    (RIT_Organization and (Job_Doctor or  Id_P1))
$ ls
pub_key  medicalData.xml.cpabe
```

4. accessing the encrypted file:

```
$ ls
pub_key Doc1_priv_key medicalData.xml.cpabe
$ cpabe-dec pub_key Doc1_priv_key medicalData.xml.cpabe
$ ls
pub_key  Doc1_priv_key  medicalData.xml
```

A.3 Preparing Web Server

Prepare the web server by downloading:

- Apache server latest version:

  ```
  $ sudo apt-get install apache2
  ```

- PHP5.* or later: some functions that were used in the system were not introduced in previous versions:

  ```
  $ sudo apt-get install php5
  ```

- configure php5 with apache2:

  ```
  $ sudo apt-get install libapache2-mod-php5
  ```

  Note: you have to restart Apache after this.

- AWS SDK needs Curl to be installed with php5, to install it:

  ```
  $ sudo apt-get install php5-curl
  ```

By installing all these libraries the server should be ready to be used as a web server. In order to be able to run the system, the user needs to use AWS PHP SDK and Azure PHP SDK. Please follow the steps in `http://aws.amazon.com/sdkforphp/` to install the latest AWS SDK PHP (Note: in the current implementation the system needs both PHP 2 and PHP 1 SDKs). In order to get Azure PHP SDK please follow the installation steps in `http://www.windowsazure.com/en-us/develop/php/` to install the SDK. **Note:** If the folder provided in the CD has been used, all the libraries for AWS and Azure are included.

A.4 Preparing the Cloud

A.4.1 AWS

- Create an account in AWS and get the authentication key and parameters into the system.
- In DynamoDB create the following tables with Hash key and Range key respectively:
  - PatientData: PatientId (String), ItemId (String).
  - PatientDataAudit: PatientId (String), ItemId (String).
  - Users: PersonId (String).
  - EmergencyReport: PatientId (String)

A.4.2 Azure

- Create an account in Azure cloud service and get the authentication key and parameters into the system.
- Create a new storage service (In the prototype case called ehr).
- In the storage service create the following tables (the system by default adds Partition key and Row key with an additional column for time stamp):
  - PersonalData
  - MedicalDataAudit

A.4.3 Authorization Files

1. AWS authentication:
   - The key file is placed */EMR/EMR/EHR.pem*
   - In *generalFunctions.php* located in */EMR/EMR/* please change the function *getAWS()* and include the following information about AWS instance:

    - key.
    - secret.
    - region.
    - ssl.certificate_authority: which is (EHR.pem)

2. Azure Authentication: change the connection string variable inside the file *RefAzure.php* located in */EMR/EMR/* to include:

    - Account Name: in prototype case EHR.
    - Acount key: can be found in Azure storage service after creating the table service.

Appendix B

Code Listing

The complete code listing is available on the attached disc.