PROJECT REPORT

on

SECURE CLOUD BASED ACCESS CONTROL AND OPTIMIZATION

Submitted in partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

by

DEVASHISH SOOD (Reg. No:1031210389)

Under the guidance of

Mrs P. VISALAKSHI

(Assistant Professor S.G., Department of Computer Science and Engineering)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM UNIVERSITY

(Under section 3 of UGC Act, 1956)

SRM Nagar, Kattankulathur- 603203

Kancheepuram District

APRIL 2016

BONAFIDE CERTIFICATE

Certified that this project report titled **"SECURE CLOUD BASED ACCESS CONTROL AND OPTIMIZATION"** is the bonafide work of **DEVASHISH SOOD (Reg.No: 1031210389)**, who carried out the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion or any other candidate.

Signature of the Guide

Mrs. P. Visalakshi
Assistant Professor (S.G.)
Department of Computer Science
and Engineering
SRM University
Kattankulathur- 603203

Signature of HOD

Dr. B. Amutha
Professor and Head of
Department of Computer Science
and Engineering
SRM University
Kattankulathur- 603203

Submitted for **Project Work Viva-voce Examination** held on_____

Place: Kattankulathur

Date:

Internal Examiner **External Examiner**

ACKNOWLEDGEMENTS

A successful project can never be prepared by the single effort or the person to whom the project is assigned, but it also demands the help and guardianship of some people who help in the undersigned actively or passively in the completion of a successful project.

I would like to thank **DR. T.R. PACHAMUTHU, Chancellor, SRM University, DR. PRABIR K BAGCHI, Vice-Chancellor, SRM University** and **DR. T.P. GANESHAN, Pro Vice Chancellor, SRM University,** for providing me the guidance to carry out the project.

It's my pleasure to thank **DR. C. MUTHAMIZCHELVAN, Director (E&T)**, of **SRM University** for giving me an opportunity to exhibit my excellence.

I am indebted to **DR. B. AMUTHA, Professor and Head of the Department of Computer Science & Engineering, SRM University** for her help and provision of infrastructure needed for completing the project.

I extend my heartfelt thanks to the respected **Project Coordinator, DR. M. MURALI** for his support and impeccable guidance.

I also thank my **Class In-charge, MRS. JACKULIN MAHARIBA** for her support and encouragement.

I express my gratitude to the honorable faculty members and my project guide **MRS. P. VISALAKSHI, Assistant Professor (S.G), Department of Computer Science & Engineering, SRM University** for her generous help, supervision and feedback.

Finally I thank the Panel and the staff of Computer Science & Engineering for their help in developing this project.

Devashish Sood

SRM UNIVERSITY
(Under section 3 of UGC Act 1956)

Department of Computer Science and Engineering
SRM University

Own Work* Declaration Form

This sheet must be filled in (each box ticked to show that the condition has been met). It must be signed and dated along with your student registration number and included with all assessments you submit – work will not be marked unless this is done

To be completed by the student for all assessments

Degree Programme : in Computer Science and Engineering

Student Name : DEVASHISH SOOD

Registration Number : 1031210389

Title of Work : Secure Cloud based Access Control and Optimization

I / We hereby certify that this assessment complies with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is my / our own except where indicated, and that I / We have met the following conditions:

- Clearly referenced / listed all sources as appropriate ☐
- Referenced and put in inverted commas all quoted text (from books, web, etc) ☐
- Given the sources of all pictures, data etc. that are not my own ☐
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present ☐
- Not sought or used the help of any external professional academic agencies for the work ☐
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources) ☐
- Complied with any other plagiarism criteria specified in the Course handbook / University website ☐

I understand that any false claim for this work will be penalised in accordance with the University policies and regulations. ☐

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referencing, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

Please Note: If you are still unsure about what plagiarism is or need advice on how to avoid it, you can

1. Consult your programme handbook
2. Refer the University website or
3. Contact any one of the following for assistance:
   Your Faculty, Course Co-ordinator, Project Supervisor, Faculty Advisor, Head of the Department.

* The University's degrees and other academic awards are given in recognition of a student's **personal achievement**. All work submitted by students for assessment is accepted on the understanding that it is the student's own effort.

** Plagiarism is defined as the submission or presentation of work, in any form, which is not one's own, without **acknowledgement of the sources (even if it's the author's previous work)**.The incorporation of material without formal and proper acknowledgement (even with no deliberate intent to cheat) can constitute plagiarism.

DOCUMENT

SCORE

93 of 100

ISSUES FOUND IN THIS TEXT

51

PLAGIARISM

7%

Contextual Spelling 10

Confused Words	10	

Grammar 3

Wrong or Missing Prepositions	3	

Punctuation 1

Comma Misuse within Clauses	1	

Sentence Structure 5

Misplaced Words or Phrases	5	

Style 32

Passive Voice Misuse	27	
Unclear Reference	3	
Improper Formatting	1	
Wordy Sentences	1	

Vocabulary enhancement

Checking disabled

ABSTRACT

We propose a decentralized access controlled scheme for secure data storage in clouds that supports public key infrastructure. In the proposed scheme, the data is transferred by public key encryption among users by the RSA algorithm. Our scheme also has the added feature of generating attributes for easily granting permission to transfer data. These attributes are sent via e-mail and can trigger an encrypted transfer of data stream, which can be decrypted only by the validated users. The cloud does not store the private keys of users and only stores public keys. The scheme prevents tampering by performing integrity checks and verifying the message digest. It works on the principles of decentralization and robustness. The computation and transfer of cipher text, and storage overheads are the same as a centralized scheme, but without the single point of failure in a decentralized scheme. The performance of the scheme has been measured by time complexity of the operations using big-O notation.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

MVC	Model View Controller
AJAX	Asynchronous Java and XML
JSP	Java Server Pages
JSTL	Java Standard Tag Library
REST	Representational State Transfer
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SHA	Secure Hash Algorithm
IAAS	Infrastructure as a Service
SAAS	Service as a Service
PAAS	Platform as a Service
WSDL	Web Services Description Language
EC2	Elastic Compute 2
EB	Elastic Beanstalk
RDS	Relational Database Service
KDC	Key Distribution Center
PKI	Public Key Infrastructure
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
TRNG	True Random Number Generator
ORM	Object Relational Mapping

LIST OF SYMBOLS

∈	Belongs to
∀	For all
$\mathbb{P}$	Abelian Group
$\mathbb{A}$	Attribute
$\mathbb{G}$	Galois Field
ø	Euler's Totient Function

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Cloud storage allows us to store our data in the cloud, which makes it accessible online from anywhere with an internet connection. Clouds are robust, dependable and provide security to data from loss by device failure, theft and migration from systems. It also makes data available irrespective of geographical location. As is the case with any new technology, the user's intentions determine whether it is used for good or evil. This means data security is a major concern as failures can lead to private data leaks, which can then be transmitted to many other sources.

1.1 Various Types of Cloud Services

The cloud architecture is determined by the service it provides such as Software as a Service (SAAS), Platform as a Service (PAAS), and Infrastructure as a Service (IAAS).

- Platform as a Service provide platforms to developers to create applications. Eg: - Heroku, Google Application Engine and Red Hat's OpenShift.
- Software as a Service provides services to non-developers for easy to use softwares. Eg:- Google Drive and Dropbox.
- Infrastructure as a Service provides the ground up access to developers to build things from scratch with the infrastructure. Eg: - Navisite, Exoscale and Softlayer.

1.2 Various Software Architectural frameworks for Clouds

There are two prevalent schemes namely:

- Service Oriented Architecture (SOA):

  It relies on service orientation as the fundamental design principle and describes how messages are parsed. SOA provides rules for communicating metadata and the like.

  It uses the Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP). It is an old architecture to provide solid ground for service providers over the internet.
- Representational State Transfer :

  The application of REST induces the measurable quantities of Performance, Scalability, Reliability and many other factors in the components of the World Wide Web like textboxes, inputs, buttons etc. The architectural constraints of REST help hypermedia and the HTTP based system to operate in a well behaved manner.

  Some of the REST constraints are Stateless, Client Server, Cacheable, Uniform interface, Layered System etc.

1.3 Benefits of the Cloud

- They provide access control to users.
- They provide network security.
- Data is encrypted on the cloud.
- They provide geographical location independence.
- They provide protection from device failure, theft and migration.

1.4 Amazon Web Services and other developer tools

AWS provides space on the cloud for free in the Services of Elastic Compute 2 (EC2), Elastic Beanstalk (EB) and S3 buckets.

EC2 provides a virtual Linux system which can be customized however the user needs to develop the cloud.

Elastic Beanstalk provides a customized system of some server like Tomcat 8 and Java 8 (or) Python and Glassfish (or) Ruby on Rails etc.

S3 buckets store data like images, videos etc. and provide URL to access them over the web.

Route 53: Provides Static IP which can be used to bind the Virtual Private Cloud.

Redis: An independent service (not linked to AWS) which provides key value hash storage for html encrypted headers which contain JSON data.

CHAPTER 2

LITERATURE REVIEW

Wang et al [1] discussed some schemes to create a secure and dependable storage model. Creation of some basic authenticating schemes has been discussed in [2]. Cryptographically securing the cloud and connections via Diffie-Hellman techniques and other Security Provider services like X.509 to manage digital certificates and Key Distribution Centers (KDC) have been discussed by [3] and improved upon greatly by Sushmita Ruj et al in [6]. Some methods to manage signatures and anonymous authentication of users, if necessary in the cloud are also discussed in [6].

2.1 Definitions of various Algorithms and Terminology

Diffie-Hellman technique: Technique used to get a secure private line of communication between two people without any eavesdroppers getting access. It generates a shared key for both ends leaving out eavesdroppers.

Key Distribution Center: A server dispensing sensitive keys which are encrypted using the X.509 standard.

X.509: It is a standard to manage Digital Certificates in the Public Key Infrastructure which recommends using various techniques and collections of algorithms to maintain quality.

SHA-1: Message Digest generating algorithm which is potentially vulnerable.

SHA-2\ SHA-256: Message Digest generating algorithm which is newer and reliable.

Message Digest: A small hashed value of constant length used to verify the integrity of the message. It detects modification of the data and attempts at tampering.

2.2 Security and Privacy

Placing critical data in the hands of a cloud provider should come with the guarantee of security and availability. Security and privacy are interlinked in the cloud. On one hand the user data should be kept private. On the other hand the cloud performs a variety of checks on the data to ensure the integrity of the cloud and to check for signs of tampering. Trust upon cloud services has been discussed by [4][5]. Privacy preserving schemes have been discussed by Ruj et al in [6]. The solution to such a scenario is to allow the cloud services to work only on encrypted data to prevent a breach of privacy. The cloud only needs to check the cryptographic checksums of the encrypted data to ensure its integrity and doesn't need to know the data itself. One way to ensure this is to limit the control given to the services and define interactions among them. An external Key Distribution Center (KDC) can be setup to manage public keys only. Encrypting and Decrypting services can have a close interaction with the KDC and data transfer among users can be done via Public Key Infrastructure. This allows us to decentralize the model and delegate responsibility to the users. Thus, the user can rely on access control of the data immune to error and hacks.

2.3 Model View Controller (MVC) architecture

MVC is a RESTful architecture which allows easy separation of service roles and authority. It allows decoupling interlinked processes and good presentation of the modules and associations.

The need for MVC depends on:

Optimization of *Data model* (Database Efficiency) – Wants to make the database as compact as possible, requiring minimum number of CPU cycles, memory transfers and increase efficiency.

Optimization of *Business model* (Revenue) – Make the data as well explained to the consumer as possible. Expand the data and customize it for every user to give them the personal touch. Make the product as tailored for the needs of the individual user based on parameters of consumer income, job and desirability.

Model: The *back end* database from the Data model which is optimized and compacted. (The Back-end is PostgreSQL, which is optimized for the web-apps and supports heavy concurrency control and minimizes DB transaction time.)

View: Refers to the expanded, customized and visually appealing *front end* which needs to be designed and personalized for consumer types from the Model. User interacts with the System via the front end.(Front-end: Set of HTML pages, JSP, CSS, JavaScript and AJAX content returned from the website to user browser.)

Controller: The program handling the mappings of the Model to View. It interacts with the View and Model to provide access control to the user. (Java-based controller with Spring Dependency Injection to provide Inversion of Control. It gets the URL and parameter headers. Processes the URL with the help of various services and returns the View Object and URL.)

CHAPTER 3

SYSTEM ANALYSIS

The basic concepts from which the system derives its roots from are explained in this section.

3.1 Decentralization:

Having a centralized data scheme is a secure node as well as a single point of failure. If the central server fails, the entire task fails, thus decentralization is essential. It involves allowing users to access the cipher texts on their systems and decrypting it offline. However, we can simplify the amount of work to be done by the clients by using attributes.

Table 3.1: Notations

Symbols	**Meanings**
U_x	User of Index X
$\mathbb{U}$	Set of Users
A_{xi}	Attribute of M_I to U_X
$\mathbb{A}$	Set of all attributes
M_I	Plaintext of index i
C_{XI}	Cipher encrypted with pu_x and having plaintext M_i
Pu_x	Public key of U_x
Pr_x	Private key of U_x

3.2 Mathematical Background

RSA uses asymmetric key encryption which assigns all users with different keys for encrypting and decrypting.

Fermat's Little Theorem: $a^p = a \pmod p$, which significantly saves time and reduces space complexity in prime number finding and primality testing by Miller Rabin's Algorithm of huge primes probabilistically.

Euler's Totient function (ø* (n)*): The cardinality count of the reduced set of residues of n. In prime numbers ø (n) = n – 1. In product of two primes p, q: ø (p x q) = (p - 1) x (q - 1)

Chinese Remainder Theorem: Which allows modulo multiplication to be split into smaller fragments and used in conjunction with Fermat's Little Theorem to encrypt data.

3.3 Public Key Encryption

The following represents the RSA algorithm developed by Rivest, Shamir and Adleman in 1978, which remains computationally complex and is easily made impossible to break by increasing bit lengths and using TRNG.

3.3.1 RSA Mathematical Model Initialization: We set the bit length as z. We select two large random primes from the *Abelian group* p, q ∈ ℙ such that |p|=|q|<=z/2, where |a| represents the digit count of the number. We use *Miller Rabin's algorithm* to find probabilistic primes of the desired length with a *Secure Random Number Generator.*

We get n = p x q, where n ∈ ℙ x ℙ.

We calculate ø (n) = (p-1)(q-1), where ø (n) is *Euler's totient function*. Then we find a number 'e' such that gcd (ø (n), e) = 1, 1< e < m.

Also calculate d = mod_Inverse (m, e) by *Extended Euclid's theorem*, which is the solution to the equation e*d = 1 mod ø (n), or $d = e^{-1}$ mod n.

Publish Pu_X (n, e) and store Pr_X (n, d).

3.3.2 Encryption via RSA: To encrypt a message m, convert to byte array and apply

$$(m^e) \% n = c$$

where c = cipher.

3.3.3 Decryption via RSA: To decrypt,

$$(c^d) \% n = message.$$

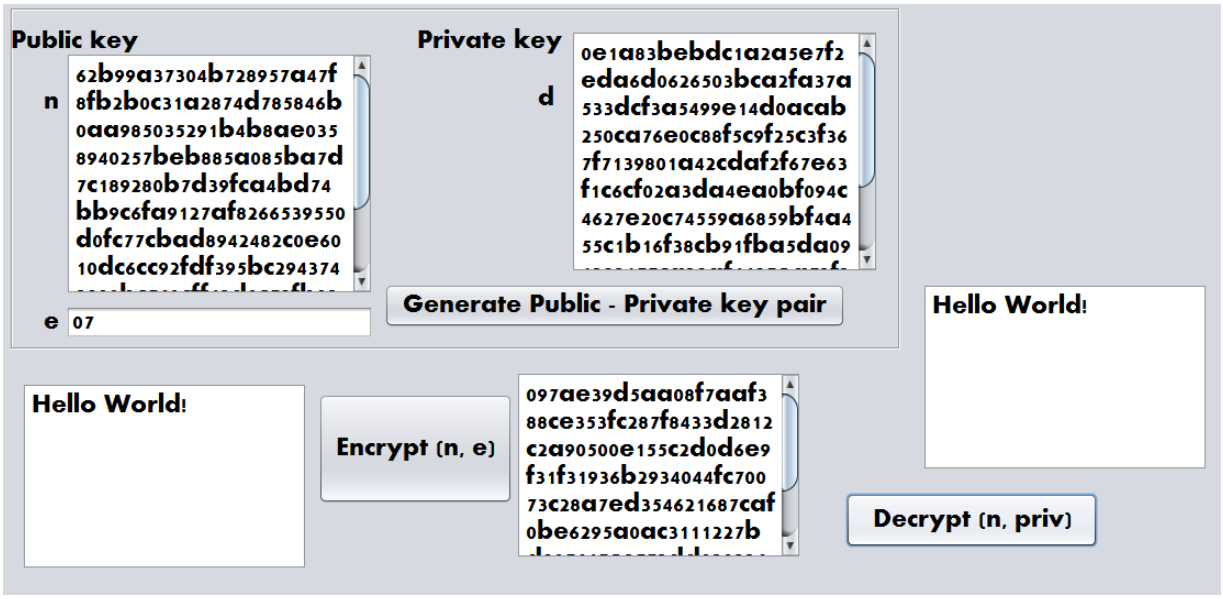


Figure 3.1 RSA

3.4 Access control:

While it is good practice to store data together and efficiency is raised by doing so, preventing unauthorized access is also necessary. All the data stored by the cloud must be encrypted by users and can be decrypted only by the users to whom the file is sent. Even the sender or the cloud cannot decrypt data encrypted with the receiver's public key.

Nowadays group permissions allow multiple users access to their collective data and this poses a problem when combined with decentralization.

Cryptography comes to the rescue by allowing multiple users to stamp their data together in a cryptographic hash so that the decentralized property remains valid. Yet integrity is not compromised by signatures which allow other users to validate the data without reading it.

Cloud systems have the encryption scheme for transferring data among users. Public key infrastructure allows users to reveal their public keys and store their private keys like a password. The cloud doesn't need access to the private keys.

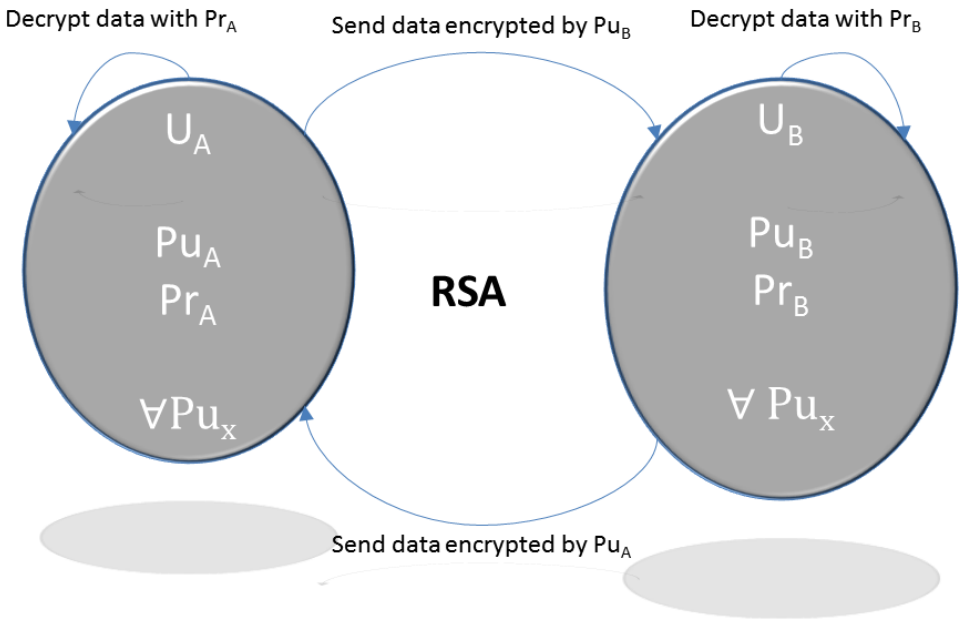


Figure 3.2 RSA communication model

CHAPTER 4

SYSTEM DESIGN

4.1 SCACO system model

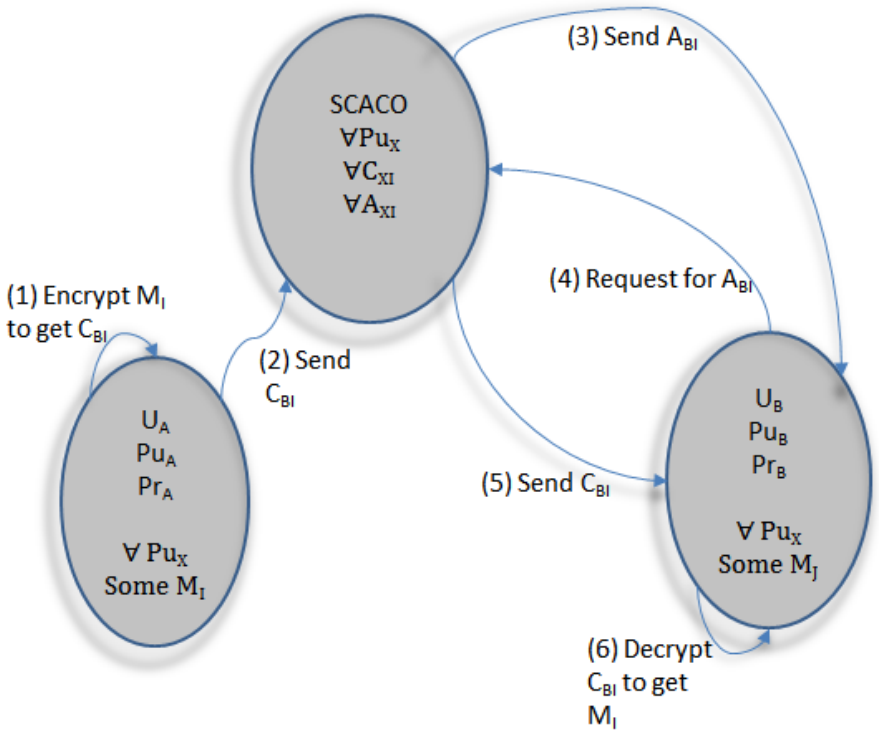


Figure 4.1

SCACO provides a cloud-based Database as a Service (DbaaS) platform which is decentralized and secured by cryptographic algorithms.

4.1.1 Cloud User Initialization:

Generate Pux, Prx, keys for users Ux ∈ 𝕌, on registration by RSA algorithm.

User A encrypts message via RSA to generate cipher,

C_{BI} = Encrypt-RSA (Pu_B, M_I).

SCACO cannot decrypt the message.

Receive Ciphers from A.

4.1.2 Server Attribute Generation:

Assume message M_I is to be sent by U_A to U_B.

Generate attribute:

A_{BI} = substring (SHA-2 (M_I, Pu_B))

Send A_{BI} to B.

The attribute should depend on the message and be recipient specific. Attribute is made small to allow easy entry by humans. We used 5 digit hexadecimal codes as attributes.

4.1.3 Server Distribution of data:

Verify (A_{BI}) ∈ 𝔸, sent by B.

Send C_{BI} to B via SSL or SSH depending on authentication requirements.

U_B can decrypt:

M_{BI} = Decrypt-RSA (Pr_B, C_{BI}).

4.2 Advantages of SCACO

- SCACO cannot decrypt the ciphers it stores.
- User can carry list of attributes whose size is much smaller and request all ciphers multiple times.
- RSA prevents modification of ciphers stored on SCACO.
- SCACO can perform integrity checks on data.

4.3 Security

RNG: Random number Generator used should abide by FIPS 140-2, Security Requirements for Cryptographic Modules and RFC 1750: Randomness Recommendations for Security. [10][11]

Discrete logarithm: The inverse problem to exponentiation is the discrete logarithm problem, which allows us to find primitive roots, solving the prime factorization problem of two huge primes. The finding of discrete logarithm is an NP-hard problem.

Some solutions find discrete logarithm in O (n log n + n $\sqrt{}$ p) time [9], which is still unfeasible to decrypt the public key of a user.

Possible approaches to attacking RSA are:

- Brute force based attacks (which are infeasible for given size of numbers)
- Mathematical attacks based on the difficulty of computing Euler's Totient Function, by factoring mod n)
- Chosen cipher text attacks from internal rogue users (given properties of RSA) [7]

All the attacks do not work and keeping them in mind we should increase the bit length with time to keep the complexity high.

4.4 Computation Complexity

K is constant bitlength.

Public key operations take O (k2) steps.

Private key operations take O (k3) steps.

Key generation takes O(k4) steps, where k is the bitlength (n).

Note: log d = Approx. (log n) and O (log e) = 1. Therefore all operations run in Approx. O (1) for small k. Approx. 35ms for each operation, k=1024. [9][10]

CHAPTER 5

CODING AND TESTING

SCACO was implemented with J2EE and Java 8 in Netbeans IDE. It utilizes the Web-app archetype Spring4 and is built with Ant into WAR file deployable on the Internet or localhost by Tomcat8 as a server. The software also uses Hibernate ORM to map SQL queries automatically. Further implementation uses JavaMail on SMTP to send attributes.

All the components can be tested by JUNIT and the integration test can be done by using Selenium.

5.1 Project Structure

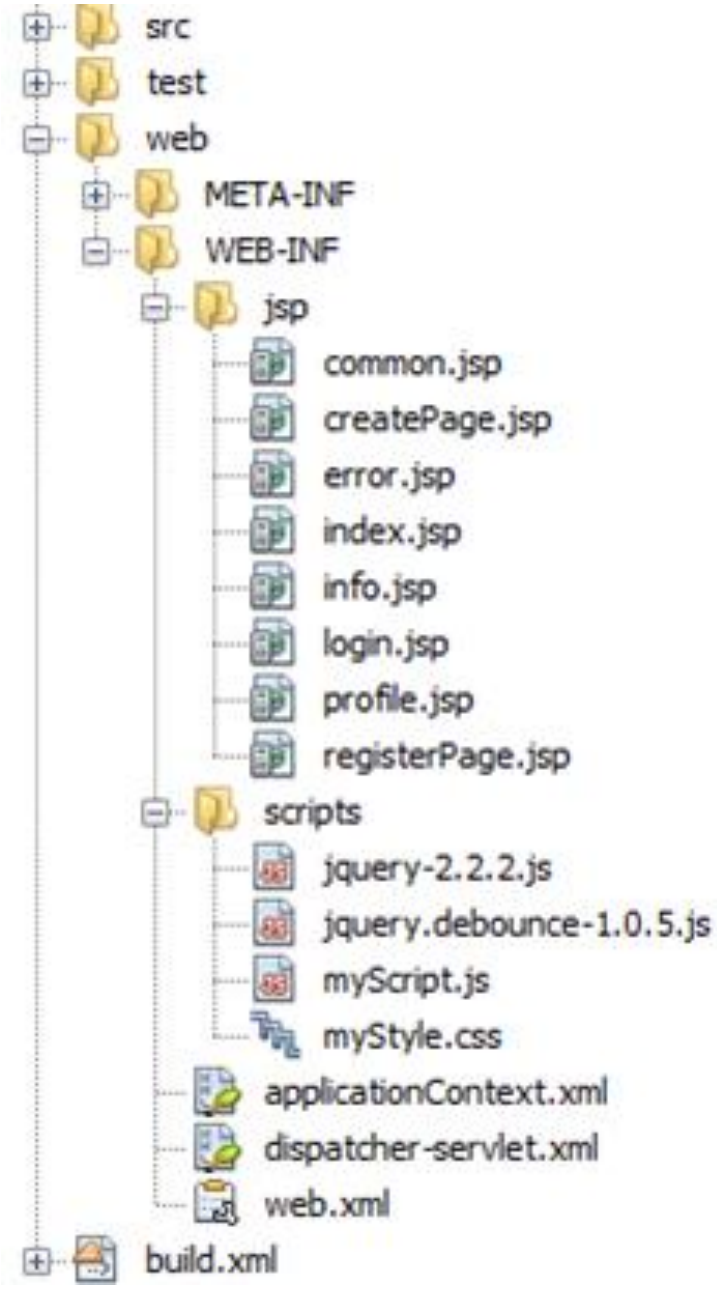


Figure 5.1

Web.xml: The project consists of Web.xml which receives the requests from the user. Web.xml contains all the dependency information of the project. Web.xml redirects to the dispatcher servlet which handles static web pages.

Dispatcher Servlet:

The dispatcher servlet gets pages from the Controller which converts the Dynamic JSP pages into dynamic HTML after server side includes. Dynamic HTML only consists of JavaScript which is executed on the Client side. Dispatcher servlet maps controller and contains the libraries for the Spring Framework.

JSP:

Contains JSTL enabled pages which call the controller to embed the necessary object into them.

Eg: - **Index.html**

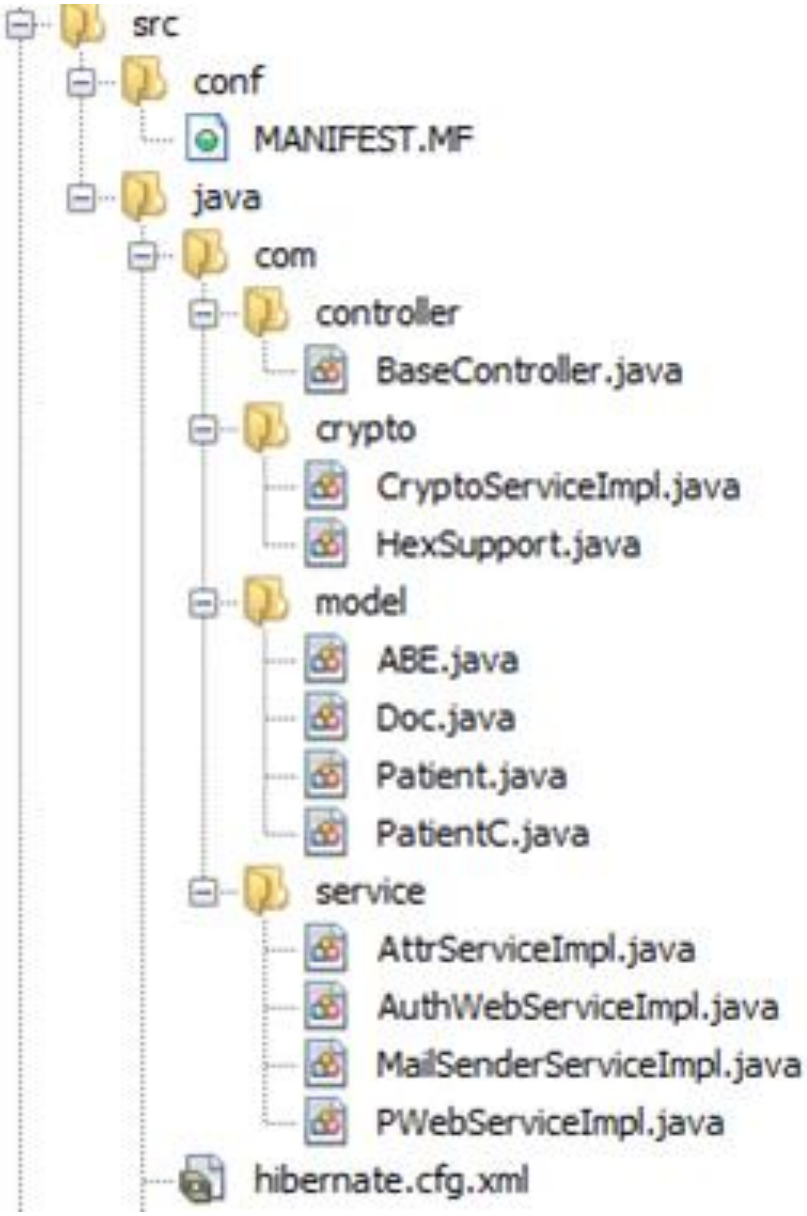
```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%
response.addHeader("Cache-Control", "no-cache,no-store,private,must-revalidate,max-stale=0,post-check=0,pre-check=0");
response.addHeader("Pragma", "no-cache");
response.addDateHeader ("Expires", 0);
%>
<!DOCTYPE HTML >
<html>
	<head>
	<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
	<title>Read Records</title>
	<link rel="stylesheet" type="text/css" href="./scripts/myStyle.css">
	</head>
	<body>
		${msg}<br>${result}<br>
		<div class="righty">
			<form id="logout" action="logout" method="post">
			<input type="hidden" name="key" value="${key}" />
```

```
            <input  value="Logout" type="submit" form="logout"/>
            </form>
</div>
<div class="righty">
            <form id="profile" action="profile" method="post">
            <input type="hidden" name="key" value="${key}" />
            <input  value="Account details" type="submit" />
            </form>
</div>
<div>
            <form id="createPage" action="createPage" method="post">
            <input type="submit" value="Create Records">
            <input type="hidden" name="key" value="${key}" />
            </form>
</div>
<div>
            <form id="deleteForm" action="delete" method="post">
            <input type="hidden" name="key" value="${key}" />
            </form>

            <form  action="deleteAll" method="post">
            <input type="submit" value="Delete All records"/>
            <input type="hidden" name="key" value="${key}" />
            </form>
</div>
<div>
            <form  action="sampleAll" method="post">
            <input disabled="true" type="hidden" value="Insert Sample Database"/>
            </form>
</div>
```

```
<div>
<form  action="common" method="post">
<input type="hidden" name="key" value="${key}"/>
<input type="submit" value="Access Common Cloud"/>
</form>
</div><br><br>
<div class="next">
${count} Records in ${readTime} <br>
<table >
<th>ID/ Delete</th><th>Name</th><th>Date-
Time</th><th>Symptoms</th><th>Diagnosis</th>
<c:forEach items="${patients}" var="patient">
<tr>
<td><input type="submit" form="deleteForm" name="id"
value=<c:out value="${patient.getId()}" />></td>
<td><c:out value="${patient.getName()}" /></td>
<td><c:out value="${patient.getDate()}" /></td>
<td><c:out value="${patient.getSymptoms()}" /></td>
<td><c:out value="${patient.getDiagnosis()}" /></td>
</tr>
</c:forEach>
</table>
</div><br><br>
</body>
</html>
```

5.2 Source Files



All the class files in modules are listed here.

5.2.1 Crypto Service

```
package com.crypto;
import java.io.IOException;
import java.math.BigInteger;
import java.security.SecureRandom;
import com.crypto.HexSupport;
public class CryptoServiceImpl {
   private BigInteger n, d, e;
   private int bitlen = 1024;
   /** Create an instance that can encrypt using someone elses public key. */
   public CryptoServiceImpl(BigInteger newn, BigInteger newe) {
      n = newn;
      e = newe;
   }
   public CryptoServiceImpl(String newn, String newe,String newd) throws IOException{
      n = stringToBig(newn);
      e = stringToBig(newe);
      d = stringToBig(newd);
   }
   /** Create an instance that can both encrypt and decrypt. */
   public CryptoServiceImpl(int bits) {
      bitlen = bits;
      generateKeys();
   }
   /** Encrypt the given plaintext message. */
   public synchronized String encrypt(String message) throws IOException{
      return bigToString((new BigInteger(message.getBytes())).modPow(e, n));
```

```
}
/** Encrypt the given plaintext message. */
public synchronized BigInteger encrypt(BigInteger message) {
   return message.modPow(e, n);
}
/** Decrypt the given ciphertext message. */
public synchronized String decrypt(String message) throws IOException{
   return new String((stringToBig(message)).modPow(d, n).toByteArray());
}
public synchronized String decrypt(String message,BigInteger n,BigInteger d) throws IOException{
   return new String(stringToBig(message).modPow(d, n).toByteArray());
}
/** Decrypt the given ciphertext message. */
public synchronized BigInteger decrypt(BigInteger message) {
   return message.modPow(d, n);
}
/** Generate a new public and private key set. */
public synchronized void generateKeys() {
   SecureRandom r = new SecureRandom();
   BigInteger p = new BigInteger(bitlen / 2, 100, r);
   BigInteger q = new BigInteger(bitlen / 2, 100, r);
   n = p.multiply(q);
   BigInteger m = (p.subtract(BigInteger.ONE)).multiply(q
         .subtract(BigInteger.ONE));
   e = new BigInteger("3");
   while (m.gcd(e).intValue() > 1) {
      e = e.add(new BigInteger("2"));
   }
   d = e.modInverse(m);
```

```
}
/** Return the modulus. */
public synchronized BigInteger getN() {
   return n;
}
/** Return the public key. */
public synchronized BigInteger getE() {
   return e;
}
public synchronized BigInteger getD() {
   return d;
}
public synchronized String bigToString(BigInteger big){

   return HexSupport.toHexFromBytes(big.toByteArray());
}
public synchronized BigInteger stringToBig(String str){
   return new BigInteger(HexSupport.toBytesFromHex(str));
}
public synchronized String[] storeKeys()throws IOException{
   String[] str=new String[3];
   str[0]=bigToString(n);
   str[1]=bigToString(e);
   str[2]=bigToString(d);
   return str;
}       }
```

5.2.2 Attribute Service

```
public class AttrServiceImpl {
    Configuration configuration = new Configuration().configure("hibernate.cfg.xml")
            .addAnnotatedClass(com.model.Doc.class)
            .addAnnotatedClass(com.model.PatientC.class)
            .addAnnotatedClass(com.model.ABE.class)
            .configure();
    StandardServiceRegistryBuilder ssrb = new StandardServiceRegistryBuilder()
            .applySettings(configuration.getProperties());
    SessionFactory sessionFactory = configuration.buildSessionFactory(ssrb.build());
public boolean validate(int fileId,String receiver,String attr){


        try{
        ABE abe=getABE(fileId,receiver);
        if(attr.equals(abe.getAttr())){
            return true;
        }
        }catch(Exception e){}
        return false;
    }
    public Patient decrypt(int fid,String attr,String rec){
        Patient pat=new Patient();
        PatientC patc=getPatientById(fid);
        ABE abe=getABE(fid, rec);
        String sender=abe.getSender();
        Doc doc=getDoc(sender);
        try{
        CryptoServiceImpl csi=new CryptoServiceImpl(doc.getPub1(),
                doc.getPub2(), doc.getPriv());
```

```
        pat.decrypt(fid, patc.getDate(),csi.decrypt(patc.getName()),
                csi.decrypt(patc.getSymptoms()), csi.decrypt(patc.getDiagnosis()));
        pat.setDoc(rec);
        }catch(Exception e){pat.setName("Error");}

        return pat;
    }
public String createAttr(int fileId,String send,String rec){
        try{
        getABE(fileId, rec);
        }catch(Exception e){
        ABE abe=new ABE();

        String attr=customMD5(getPatientById(fileId).getName().substring(0,3)
                +(System.nanoTime()%10) )
                .substring(0,5);
        abe.setAttr(attr);

        abe.setFileId(fileId);
        abe.setReceiver(rec);
        abe.setSender(send);

        saveABE(abe);

        return attr;
        }
        return "Error the attribute has been sent before.";
    }
```

5.3 Modules and Services

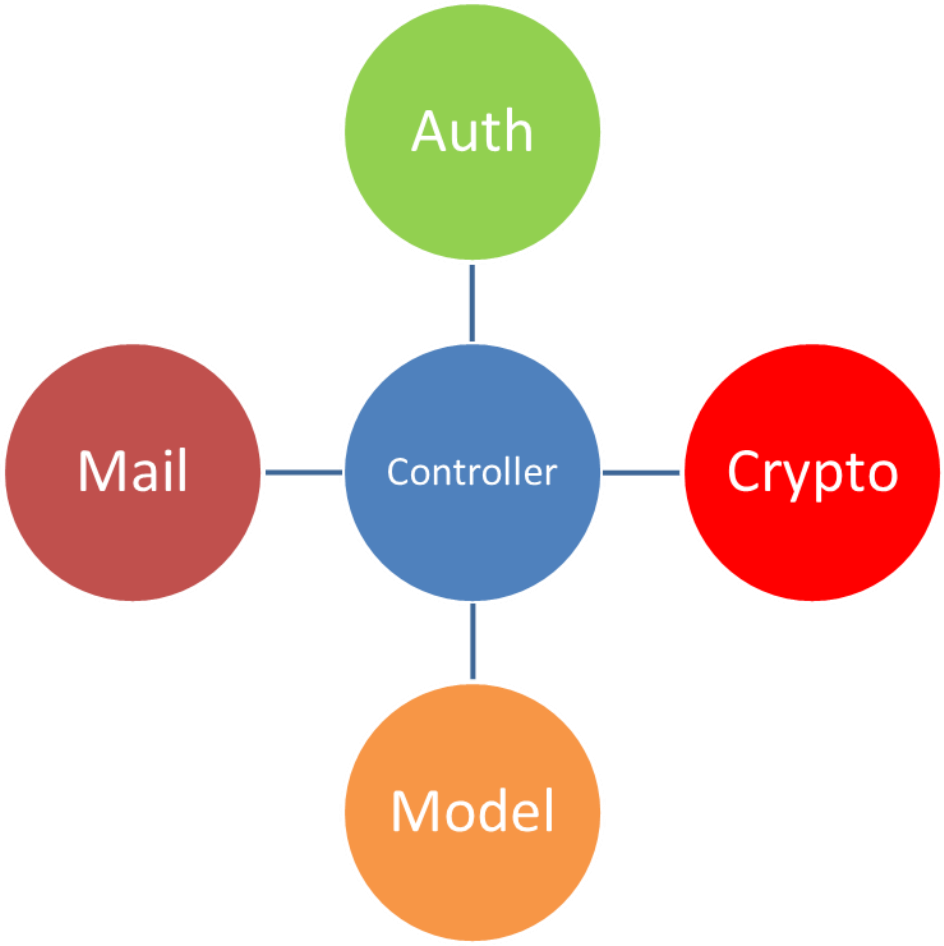


Figure 5.3

The 4 modules are:

1. Authentication Services
2. Mail Services
3. Cryptographic Services
4. Model Services

5.3.1 Authentication Service

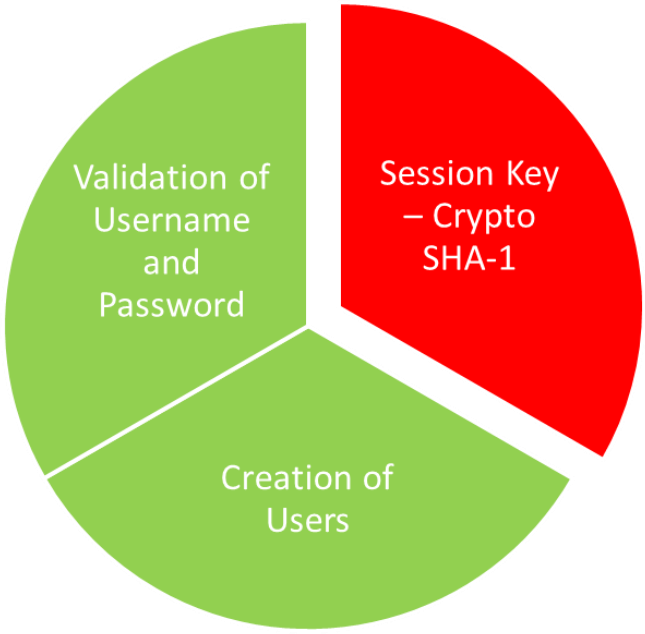


Figure 5.4

Three services namely:

1. Creation of Users
2. Validate Username and Password
3. Generate the temporary session key.

5.3.2 Crypto Service

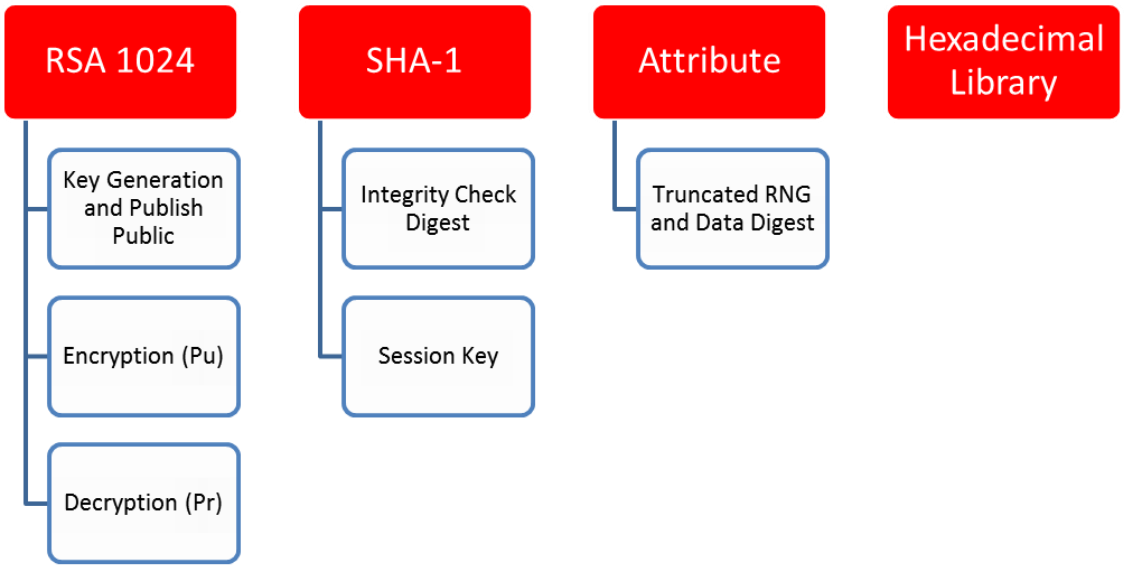


Figure 5.5

Random Number Generators can be a security risk. Much research has to be done in True Random Number Generation with sensors in computers.

5.3.3 ORM Service

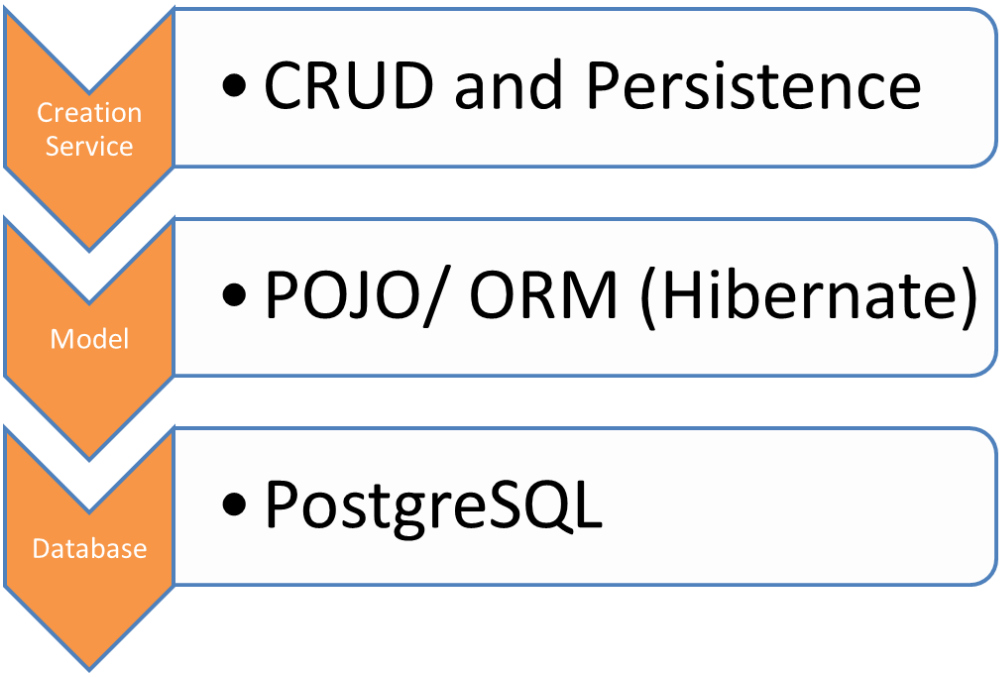


Figure 5.6

Plain Old Java Objects are initialized with the constructor in Java. They exist only during runtime and consume RAM. ORM maps the POJO to SQL and stores it in Database. It also recalls the POJO from the database when requested. Yet, the best feature of hibernate is that it can scan through POJO with SQL efficiency.

5.4 Build and Deploy

SCACO was built with Ant builder in Netbeans 8. The packaging type is WAR.

It is deployable on a variety of servers like Tomcat 8, Glassfish 4 etc.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!-- was: <?xml version="1.0" encoding="UTF-8"?> -->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans htt
       http://www.springframework.org/schema/aop http://www.springframewo
       http://www.springframework.org/schema/tx http://www.springframewor
       http://www.springframework.org/schema/context http://www.springfra
       http://www.springframework.org/schema/mvc http://www.springframewo

    <context:component-scan base-package="com.controller" />
    <mvc:annotation-driven />

    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceVie
          p:prefix="/WEB-INF/jsp/"
          p:suffix=".jsp" />
```

Figure 5.7

The following snapshots of the project on localhost show its working.

Figure 5.8 Account details

The RSA public keys of all users are available here. Only our Private key and the temporary session key are visible to us.

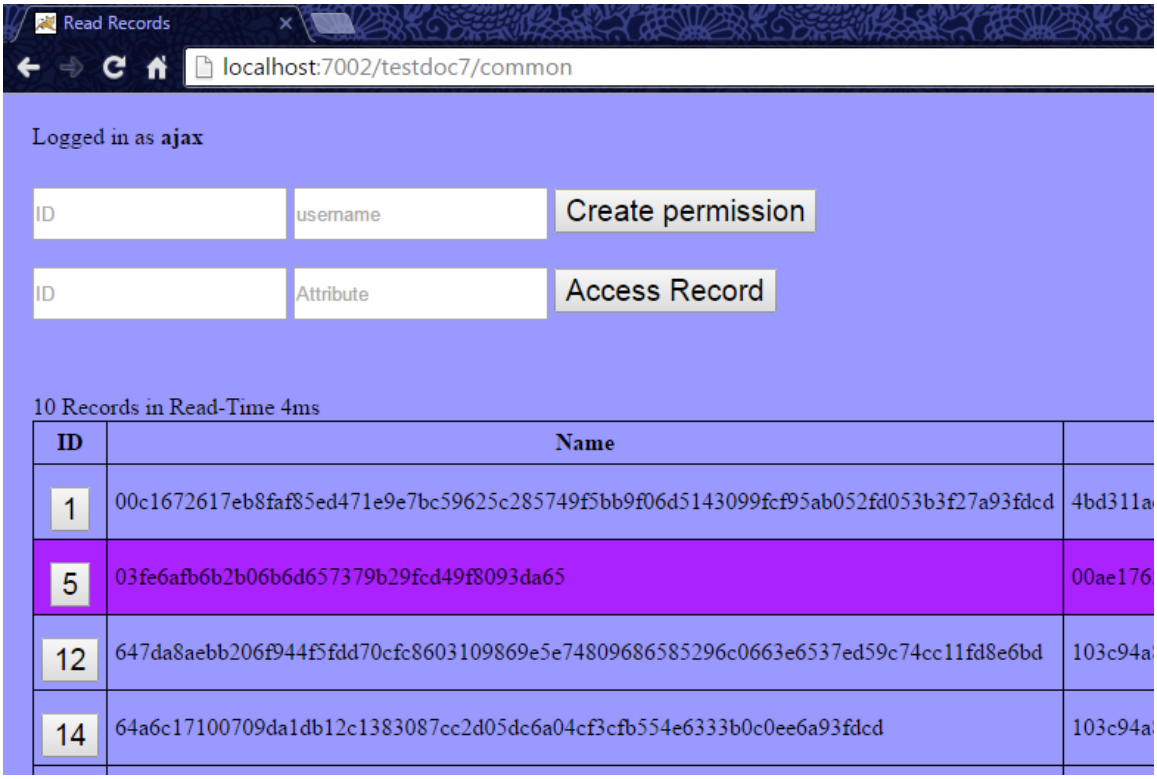


Figure 5.9 Attribute creation and usage

The transfer of cipher text on human readable attributes simplifies tasks from the common cloud.

Equations for Attribute generation are mentioned in Section 4.1.

Figure 5.10 Login and Authentication

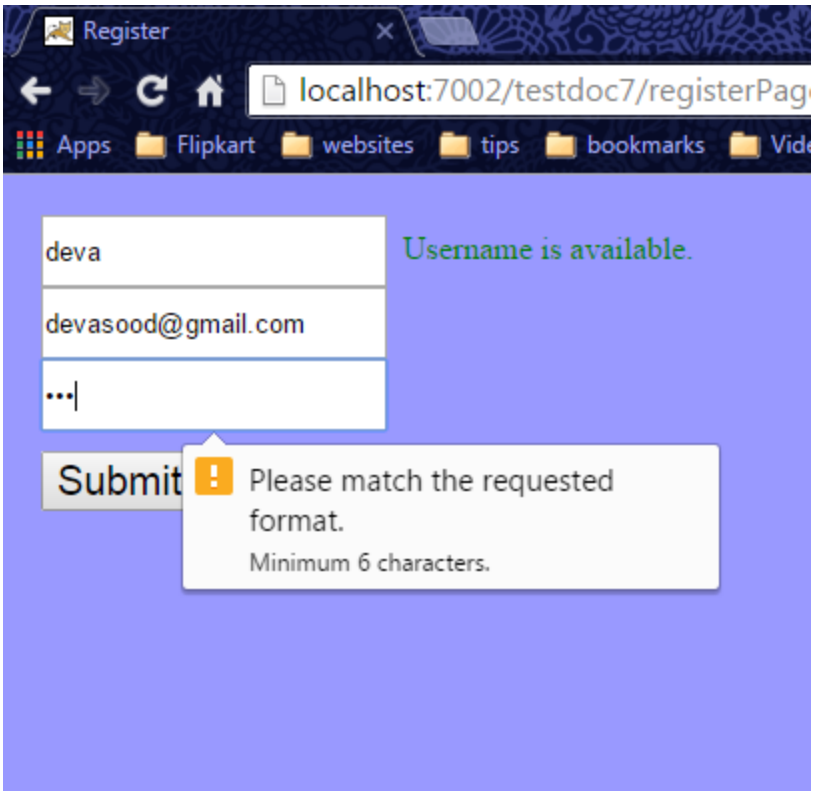


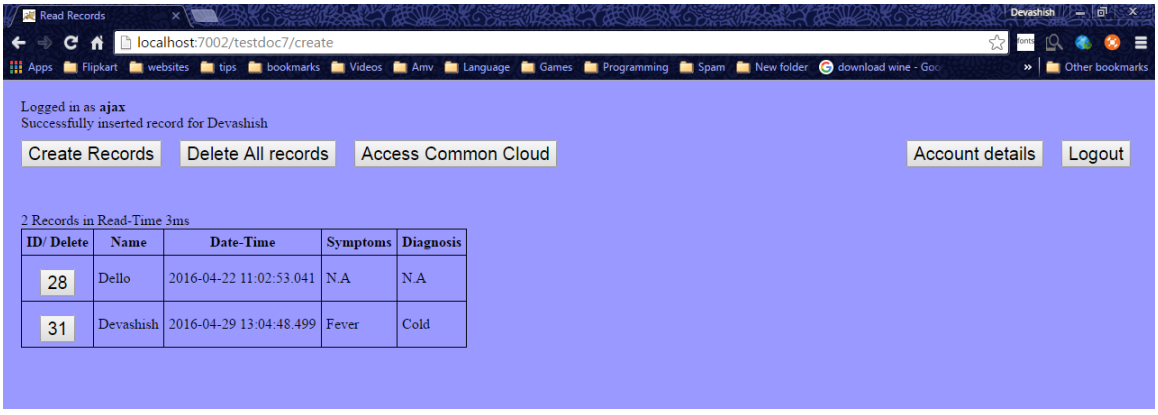
Figure 5.11 Validation on both Client Side and Server Side

The username availability is validated by Server Side via Ajax request and Key Debouncing. The pattern matching JavaScript validation is on the Client Side.

Creation

Deletion

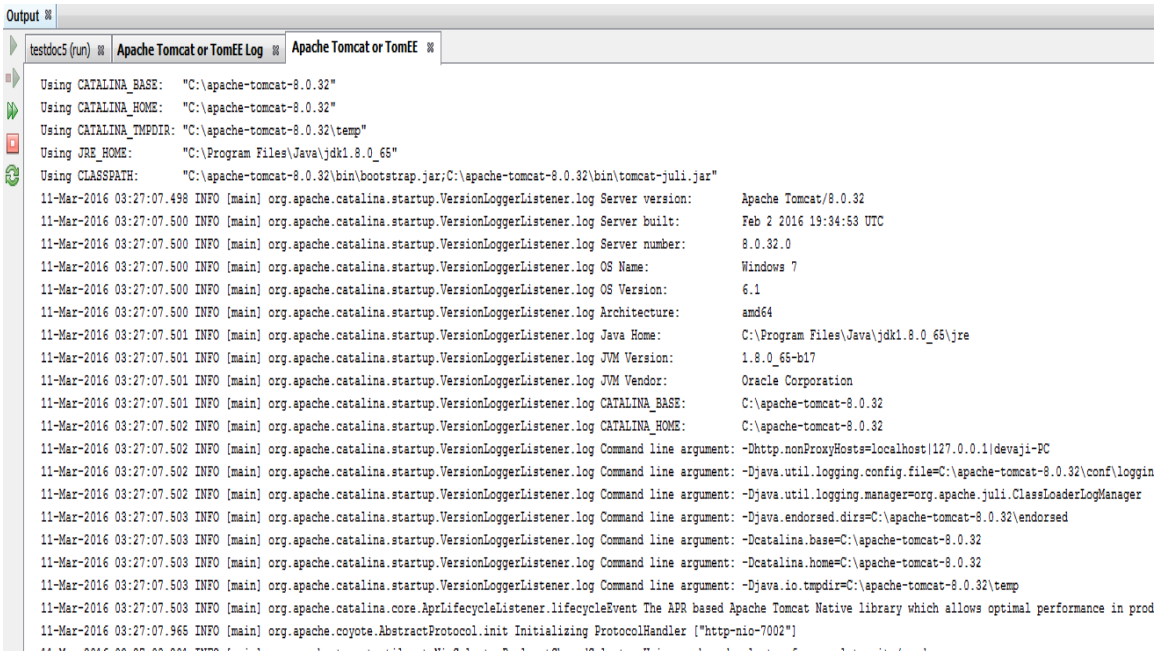
Read

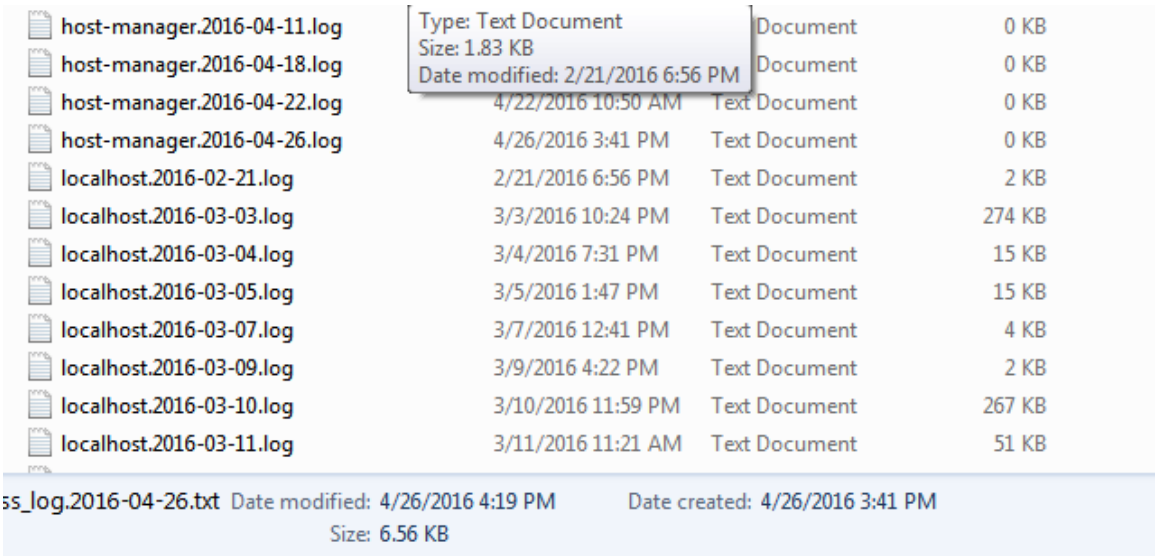


Figures 5.12

5.5 Test Logs

Figure 5.13





List of all logs executed on the system. It has times of every action and command along with errors which occurred. It even has an SQL log of all transactions.

CHAPTER 6

CONCLUSION

We have implemented a Secure Cloud-based Access Control and Optimization system based on Public Key Infrastructure and the Model View Controller architecture. The keys are asymmetric in the public key model. The public key infrastructure allows us to store public keys from the user and distribute them securely to other users of the cloud. The users can generate and store their own private keys or use the system generated private keys.

The cloud is decentralized into User Clouds with RSA keys. Every user can store their data offline on their systems and keep an encrypted copy on the common cloud. This serves as a way to restore data in case of user system failure/ migration.

Attributes are generated to provide permissions to Users from the common clouds. Attributes are user and file specific, meaning Attributes for one user cannot be used by other users to access the files.

Integrity checks are performed to prevent tampering by Message Digests (SHA-2). Session keys are generated on every login to validate users through HTTP post and revoked on logout or expiry.

The system was implemented successfully on the Web-EE archetype of J2EE with JRE 8 and Netbeans IDE. We also used Hibernate ORM to map

Plain Old Java Objects to SQL. The system was built as a WAR file and run on Tomcat 8.

Custom RSA-1024 was implemented from scratch with BigIntegers, and is extensible to any number of bits like 4096 or 8192.

The services were compared to the libraries from the Legion of the Bouncy Castle, a security provider for all platforms. All encryption and decryption was successfully completed in Constant Time Approx. O (1).

For the purposes of this project the Model theme was based on sensitive medical patient record transfers among Doctors. The project was even successfully tested on AWS-ElasticBeanstalk and Relational Database Service (RDS) and is scalable and extensible. The ElasticBeanstalk sets up an environment with JDK 8 and Tomcat 8 to be used for deploying files. The RDS sets up with username and password along with access keys and provisions for a Virtual Private Cloud.

CHAPTER 7

FUTURE ENHANCEMENT

We would like to research Multi Party Signatures, Load Balancing and Content Distribution Networks of the Cloud and include such functionality in the future. One limitation of this model is that Multiparty Signatures are not generated by the same mathematical equation, rather divided into individual attributes. Also some measures to protect the cloud from Denial of Service were mostly unexplored.

Multi Party Signatures allow the verification of data by other users with their keys.

Load balancing distributes the load of a server by making copies of it which concurrently exist and are merged.

Content Distribution networks bring the data closer geographically by caching in their servers.

Denial of Service and Distributed Denial of Service spams the cloud with requests. Using human verification before computation intensive tasks helps prevent such spam.

We would like to provide such future enhancements in SCACO.

APPENDIX A

Base Controller

```
package com.controller;


import com.model.Doc;
import com.service.AttrServiceImpl;
import com.service.AuthWebServiceImpl;
import com.service.MailSenderServiceImpl;
import com.service.PWebServiceImpl;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;


@Controller
public class BaseController {
    AuthWebServiceImpl awsi=new AuthWebServiceImpl();
    PWebServiceImpl pwsi =new PWebServiceImpl();
```

```
AttrServiceImpl asi=new AttrServiceImpl();

@RequestMapping(value = {"/"},method = RequestMethod.GET)
public String login(ModelMap model){
   model.addAttribute("msg","Login");
   return "login";
}
@RequestMapping(value = {"/logout"},method = RequestMethod.POST)
public String logout(ModelMap model,
      @RequestParam(value = "key") String key
      ){
   model.addAttribute("msg",awsi.invalidateKey(key));
   return "login";
}

@RequestMapping(value = {"/profile"},method = RequestMethod.POST)
public String profile(ModelMap model,
      @RequestParam(value = "key") String key
      ){
   String b="<b>",be="</b>";

   Doc doc=asi.getDoc(awsi.getDoc(key));
   model.addAttribute("username",b+doc.getUsername()+be);
   model.addAttribute("n",b+doc.getPub1()+be);
   model.addAttribute("e",b+doc.getPub2()+be);
```

```
        model.addAttribute("d",b+doc.getPriv()+be);
        model.addAttribute("key",key);
        model.addAttribute("user",awsi.getDoc(key));
        model.addAttribute("docs",awsi.getDocs());
        return "profile";
    }

    @RequestMapping(value = "/login",method = RequestMethod.POST)
    public String index(ModelMap model,
            @RequestParam(value = "user") String user,
            @RequestParam(value = "pass") String pass,
            @RequestParam(value = "key",required = false,defaultValue = "") String key
            ){
        boolean validate=awsi.trySess(user, key);
        if(awsi.tryLogin(user, pass)||validate){
            model.addAttribute("msg","Logged in as <b>"+user+"</b>");
            model.addAttribute("key",awsi.assignKey(user));
            try{
            long time=System.currentTimeMillis();
            List list=pwsi.getPatients(user);
            model.addAttribute("patients",list);
            model.addAttribute("readTime","Read-Time "+(System.currentTimeMillis()-
time)+"ms");
            model.addAttribute("count",list.size());
            }catch(Exception e){}
```

```
        return "index";
    }
    if(validate)
        model.addAttribute("msg","You have logged out");
    else
        model.addAttribute("msg","Incorrect username or password!");
    return "login";
}
@RequestMapping(value = "/createPage",method = RequestMethod.POST)
public String create(ModelMap model,
        @RequestParam(value = "key") String key
        ){
    model.addAttribute("date",new Date());
    model.addAttribute("user",awsi.getDoc(key));
    model.addAttribute("pass","pass");
    model.addAttribute("key",key);
    return "createPage";
}
@RequestMapping(value = "/create",method = RequestMethod.POST)
public String create(ModelMap model,
        @RequestParam(value = "name") String name,
        @RequestParam(value = "sym",defaultValue = "N.A") String sym,
        @RequestParam(value = "dia",defaultValue = "N.A") String dia,
        @RequestParam(value = "key") String sessKey
){
```

```
    try{
    pwsi.createTPatient(name,sym,dia,awsi.getDoc(sessKey));
    model.addAttribute("result","Successfully inserted record for " +name);
    }catch(Exception e){
        model.addAttribute("result","Error inserting value."+e);
        e.printStackTrace();
    }
    return index(model,awsi.getDoc(sessKey),"",sessKey);
}
@RequestMapping(value = "/delete",method = RequestMethod.POST)
public String delete(ModelMap model,
        @RequestParam(value = "id") int id,
        @RequestParam(value = "key") String sessKey
){
    try{
    pwsi.deleteTPatient(id);
    model.addAttribute("result","Successfully deleted record " +id);
    }catch(Exception e){model.addAttribute("result","Error deleting value.");}
  return index(model,awsi.getDoc(sessKey),"",sessKey);
}
@RequestMapping(value = "/deleteAll",method = RequestMethod.POST)
public String deleteAll(ModelMap model,
        @RequestParam(value = "key") String sessKey
){
    try{
```

```
        long time=System.currentTimeMillis();
    pwsi.deleteAllTPatient();
    model.addAttribute("result","Successfully deleted records\n"+
            "Delete-Time "+(System.currentTimeMillis()-time)+"ms");
    }catch(Exception e){model.addAttribute("result","Error deleting.");}
    return index(model,awsi.getDoc(sessKey),"",sessKey);
}

@RequestMapping(value = "/sampleAll",method = RequestMethod.POST)
public String sampleRecords(ModelMap model) throws Exception{
    pwsi.insertSampleData();
    model.addAttribute("msg","Sample Values Inserted");
    model.addAttribute("user","user");
    model.addAttribute("pass","pass");
    return "result";
}

@RequestMapping(value="/registerPage",method=RequestMethod.GET)
public String registerPage(ModelMap model){
    return "registerPage";
}

@RequestMapping(value="/register",method=RequestMethod.POST)
public String register(ModelMap model,
        @RequestParam(value = "user") String user,
```

```
        @RequestParam(value = "email") String email,
        @RequestParam(value = "pass") String pass
        )throws IOException{
    awsi.createDoc(user,email,pass);
    return "login";
}

@RequestMapping(value="/isUserValid",method=RequestMethod.POST)
public String isUserValid(ModelMap model,
        @RequestParam(value = "user") String user
        )throws IOException{
    model.addAttribute("info",""+awsi.isUserExist(user));
    return "info";
}

@RequestMapping(value = "common",method = RequestMethod.POST)
public String common(ModelMap model,
        @RequestParam(value = "key") String key
        ){
    if(!key.equals("Anonymous")){
        String user=awsi.getDoc(key);
        model.addAttribute("msg","Logged in as <b>"+user+"</b>");
        model.addAttribute("user",user);
        model.addAttribute("key",key);
    }
```

```
        try{
            long time=System.currentTimeMillis();
            List list=pwsi.getPatientC();
            model.addAttribute("patients",list);
            model.addAttribute("readTime","Read-Time "+(System.currentTimeMillis()-
time)+"ms");
            model.addAttribute("count",list.size());
            }catch(Exception e){e.printStackTrace();}
        model.addAttribute(model);
        return "common";
    }

    @RequestMapping(value = "decrypt",method = RequestMethod.POST)
    public String decrypt(ModelMap model,
            @RequestParam(value = "key") String key,
            @RequestParam(value = "attr") String attr,
            @RequestParam(value = "fid") int fid
            ){

        if(asi.validate(fid, awsi.getDoc(key), attr)){
            pwsi.savePatient(asi.decrypt(fid,attr,awsi.getDoc(key)));
            model.addAttribute("result","Decrypted value has been inserted.");
            return common(model, key);
        }else{
            model.addAttribute("result","Invalid attribute.");
```

```
        return common(model, key);
    }
}

@RequestMapping(value = "send",method = RequestMethod.POST)
public String send(ModelMap model,
        @RequestParam(value = "key") String key,
        @RequestParam(value = "fid") int fid,
        @RequestParam(value = "rec") String rec
        ){
    String user=awsi.getDoc(key);
    if(rec.equals(user)){
        model.addAttribute("result","Can't send file to yourself.");
    }else if(awsi.isPatientExist(user,fid)){
        try{
            String attr=asi.createAttr(fid,user, rec);
            String email[]=new String[1];
            Doc doc=asi.getDoc(rec);
            email[0]=doc.getEmail();
            MailSenderServiceImpl.sendMail(email, "New File Received","Id is "+fid+"\n"
                    + "Attribute is " + attr);
            model.addAttribute("result","Successfully sent mail to "+rec+" ."+attr);
        }catch(Exception e){model.addAttribute("result","Invalid username.");}
    }else{
        model.addAttribute("result","You don't have permission for the file.");
```

```
        }
        return common(model, key);
    }

    @RequestMapping(value = {"/error"},method = RequestMethod.GET)
    public String error(ModelMap model){
        return "error";
    }
}
```

MyScript.js

```
$(document).ready(function(){
   $("#user").keyup($.debounce(function(){
      $.post("isUserValid",
      {
        user: $("#user").val()
      },
      function(data,status){
         if(data==="true"){
            $("#validUser").html("* Username has been taken.").addClass("red");
         }else{
            $("#validUser").html("Username is available.").addClass("green");
         }
      });
   },2000));
});
```

MyStyle.css

```
body{
    background-color: #9999ff;
    margin-left: 20px;
    margin-top: 20px;
}
.long-text{
    width: 200px;
}
.lefty{
    float: left;
}
.righty{
    float: right;
}
.clear{
    float: left;
    clear: both;
    display:inline-block;
}
.red{
    color: red;
}
.green{
    color: green;
}
input{
    margin:auto;
    height: 30px;
}
input[type=submit]{
    font-size: 20px;
    margin:auto;
    margin-top: 10px;
}
td input{
    display: block;
}
table, th, td {
    padding: 5px;
    border-collapse: collapse;
    border: 1px solid black;
}
tr:hover {
    background-color: #aa22ff
}

div{
    float: left;
    margin-right: 20px;
}
div.next{
    margin-top: 50px;
    clear: both;
}
.wrapit{
    width:1000px;
    word-wrap: break-word;
}
```

RegisterPage.jsp

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!DOCTYPE HTML >

<html>    <head>

      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

      <script  type="text/javascript" src="./scripts/jquery-2.2.2.js"></script>

      <script  type="text/javascript" src="./scripts/jquery.debounce-1.0.5.js"></script>

      <title>Register</title>        <link rel="stylesheet" type="text/css"
href="./scripts/myStyle.css">

   </head>

   <body>

      <script  type="text/javascript" src="./scripts/myScript.js"></script>

      <div class="lefty">

         <form id="details" action="register" method="post">

            <input type="text" name ="user" id="user" placeholder="username"
required="true"/>  <a id="validUser"></a> <br>

            <input type="email" name="email" id="email" placeholder="email@gmail.com"
required="true"/><br>

            <input type="password" name="pass" id="pass" pattern=".{6,}"
placeholder="password" required="true" title="Minimum 6 characters."/><br>

            <input type="submit" id="register" id="register" required="true"/>

         </form>

      </div>

   </body>

</html>
```

PWebServiceImpl

```
package com.service;


import com.crypto.CryptoServiceImpl;
import com.model.Doc;
import com.model.Patient;
import com.model.PatientC;
import java.io.File;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.List;
import java.util.NoSuchElementException;
import java.util.Scanner;
import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.criterion.Restrictions;
/**
 *
```

```
 * @author devaji
 */
public class PWebServiceImpl {
    Configuration configuration = new Configuration().configure("hibernate.cfg.xml")
            .addAnnotatedClass(com.model.Patient.class)
            .addAnnotatedClass(com.model.PatientC.class)
            .addAnnotatedClass(com.model.Doc.class)
            .configure();
    StandardServiceRegistryBuilder ssrb = new StandardServiceRegistryBuilder()
            .applySettings(configuration.getProperties());
    SessionFactory sessionFactory = configuration.buildSessionFactory(ssrb.build());

    public void createTPatient(String name,String sym,String dia,String doc){

        Patient user=new Patient();
        user.setName(name);
        user.setSymptoms(sym);
        user.setDiagnosis(dia);
        user.setDate(new Date());
        user.setDoc(doc);

        savePatient(user);

        Session session=sessionFactory.openSession();
        Doc docObj=(Doc)session.get(Doc.class, doc);
```

```
    String[] keys=docObj.getKeys();

    session.close();

    PatientC userc=new PatientC();
    try{
        CryptoServiceImpl csi=new CryptoServiceImpl(keys[0],keys[1],keys[2]);
        userc.encrypt(user.getId(),user.getDate(),
                csi.encrypt(name),csi.encrypt(sym),csi.encrypt(dia));

    }catch(IOException e){e.printStackTrace();}

    savePatient(userc);

}

public void deleteTPatient(int id){

    Session session=sessionFactory.openSession();
    session.beginTransaction();

    Patient user=session.get(Patient.class, id);
    session.delete(user);

    session.getTransaction().commit();
```

```
    session.close();
}

public void deleteAllTPatient(){

    Session session=sessionFactory.openSession();
    session.beginTransaction();

    Criteria crit=session.createCriteria(Patient.class);
    List<Patient> list=crit.list();
    for(Patient patient:list)
    session.delete(patient);

    session.getTransaction().commit();
    session.close();
}

public List getPatients(String doc){
    Session session=sessionFactory.openSession();
    Criteria criteria=session.createCriteria(Patient.class)
            .add(Restrictions.eq("doc",doc));
    List list=criteria.list();
    session.close();
    return list;
}
```

```
public List getPatientC(){
    Session session=sessionFactory.openSession();
    Criteria criteria=session.createCriteria(PatientC.class);
    List list=criteria.list();
    session.close();
    return list;
}

public String trimUpper(String line){
    return line.trim().toUpperCase();
}

public String[] trimUpper(String line[]){
    for(int i=0;i<line.length;i++){
        line[i]=trimUpper(line[i]);
    }
    return line;
}

public String removeHashedOldRecordNo(String temp){
    if(temp.contains("#")){
        temp=temp.split("#")[0].trim();
    }
    return temp;
```

```
}

public void savePatient(Patient pat){
    Session session=sessionFactory.openSession();
    session.beginTransaction();

    session.save(pat);

    session.getTransaction().commit();
    session.close();
}
public void savePatient(PatientC pat){
    Session session=sessionFactory.openSession();
    session.beginTransaction();

    session.save(pat);

    session.getTransaction().commit();
    session.close();
}

public void insertSampleData() throws Exception{
    int min=0,max=Integer.MAX_VALUE;
```

```
Scanner s=new
Scanner(getClass().getClassLoader().getResourceAsStream("records.csv"));
        try{
        while(true){
           String line[]=s.nextLine().split(",;,;,",-1);
           line=trimUpper(line);
           int id=-1;
           try{id=Integer.parseInt(line[0]);}catch(Exception e){}
           if(id>=min&&id<=max){
              if(line[11].trim().length()>0){
              Patient patient=new Patient();
              patient.setName(removeHashedOldRecordNo(line[2]));
              patient.setDate(new SimpleDateFormat("dd MMM yyyy").parse(line[1]));
              patient.setSymptoms(line[9]);
              patient.setDiagnosis(line[11]);
              savePatient(patient);
              }
           }
        }
        }catch(NoSuchElementException e){
        }
    }
}
```

REFERENCES

[1] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, "Toward Secure and Dependable Storage Services in Cloud Computing," IEEE Trans. Services Computing, vol. 5, no. 2, pp. 220-232, Apr.-June 2012.

[2] S. Kamara and K. Lauter, "Cryptographic Cloud Storage," Proc. 14th Int'l Conf. Financial Cryptography and Data Security, pp. 136-149, 2010.

[3] H. Li, Y. Dai, L. Tian, and H. Yang, "Identity-Based Authentication for Cloud Computing," Proc. First Int'l Conf. Cloud Computing (CloudCom), pp. 157-166, 2009.

[4] A.-R. Sadeghi, T. Schneider, and M. Winandy, "Token-Based Cloud Computing," Proc. Third Int'l Conf. Trust and Trustworthy Computing (TRUST), pp. 417-429, 2010.

[5] R.K.L. Ko, P. Jagadpramana, M. Mowbray, S. Pearson, M. Kirchberg, Q. Liang, and B.S. Lee, "Trustcloud: A Framework for Accountability and Trust in Cloud Computing," HP Technical Report HPL-2011-38, http://www.hpl.hp.com/techreports/2011/HPL-2011-38.html, 2013.

[6] Sushmita Ruj, Milos Stojmenovic, Amiya Nayak, "Decentralized Access Control with Anonymous Authentication of Data Stored in Clouds" IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 25, NO. 2, FEBRUARY 2014.

[7] William Stallings, "Cryptography and Network Security" 4th Edition.

[8] R.L. Rivest, A. Shamir, and L. Adleman "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems" MIT, 1978.

[9] http://crypto.stackexchange.com/

[10] http://bouncycastle.org/ Libraries for implementing security providers

[11] http://security.stackexchange.com/

[12] http://www.wikipedia.com/ Encyclopedia

[13] http://google.com/ Search Engine