**SOURCE CODE**

**EncryptorAesGcm.java**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package com.sddpa.crypto;

/\*\*

\*

\* @author Ramu Maloth

\*/

import javax.crypto.Cipher;

import javax.crypto.SecretKey;

import javax.crypto.spec.GCMParameterSpec;

import java.nio.ByteBuffer;

import java.nio.charset.Charset;

import java.nio.charset.StandardCharsets;

/\*\*

\* AES-GCM inputs - 12 bytes IV, need the same IV and secret keys for encryption and decryption.

\* <p>

\* The output consist of iv, encrypted content, and auth tag in the following format:

\* output = byte[] {i i i c c c c c c ...}

\* <p>

\* i = IV bytes

\* c = content bytes (encrypted content, auth tag)

\*/

public class EncryptorAesGcm {

private static final String ENCRYPT\_ALGO = "AES/GCM/NoPadding";

private static final int TAG\_LENGTH\_BIT = 128;

private static final int IV\_LENGTH\_BYTE = 12;

private static final int AES\_KEY\_BIT = 128 ; //256;

private static final Charset UTF\_8 = StandardCharsets.UTF\_8;

// AES-GCM needs GCMParameterSpec

public static byte[] encrypt(byte[] pText, SecretKey secret, byte[] iv) throws Exception {

Cipher cipher = Cipher.getInstance(ENCRYPT\_ALGO);

cipher.init(Cipher.ENCRYPT\_MODE, secret, new GCMParameterSpec(TAG\_LENGTH\_BIT, iv));

byte[] encryptedText = cipher.doFinal(pText);

return encryptedText;

}

// prefix IV length + IV bytes to cipher text

public static byte[] encryptWithPrefixIV(byte[] pText, SecretKey secret, byte[] iv) throws Exception {

byte[] cipherText = encrypt(pText, secret, iv);

byte[] cipherTextWithIv = ByteBuffer.allocate(iv.length + cipherText.length)

.put(iv)

.put(cipherText)

.array();

return cipherTextWithIv;

}

public static String decrypt(byte[] cText, SecretKey secret, byte[] iv) throws Exception {

Cipher cipher = Cipher.getInstance(ENCRYPT\_ALGO);

cipher.init(Cipher.DECRYPT\_MODE, secret, new GCMParameterSpec(TAG\_LENGTH\_BIT, iv));

//System.out.println("c\_Text is "+cText);

byte[] plainText = cipher.doFinal(cText);

return new String(plainText, UTF\_8);

}

public static String decryptWithPrefixIV(byte[] cText, SecretKey secret) throws Exception {

ByteBuffer bb = ByteBuffer.wrap(cText);

byte[] iv = new byte[IV\_LENGTH\_BYTE];

bb.get(iv);

//bb.get(iv, 0, iv.length);

byte[] cipherText = new byte[bb.remaining()];

bb.get(cipherText);

//System.out.println("Secret Key "+secret.toString());

String plainText = decrypt(cipherText, secret, iv);

return plainText;

}

public static void main(String[] args) throws Exception {

String OUTPUT\_FORMAT = "%-30s:%s";

String pText = "Hello World AES-GCM, Welcome to Cryptography!";

// encrypt and decrypt need the same key.

// get AES 256 bits (32 bytes) key

SecretKey secretKey = CryptoUtils.getAESKey(AES\_KEY\_BIT);

// encrypt and decrypt need the same IV.

// AES-GCM needs IV 96-bit (12 bytes)

byte[] iv = CryptoUtils.getRandomNonce(IV\_LENGTH\_BYTE);

byte[] encryptedText = EncryptorAesGcm.encryptWithPrefixIV(pText.getBytes(UTF\_8), secretKey, iv);

System.out.println("\n------ AES GCM Encryption ------");

System.out.println(String.format(OUTPUT\_FORMAT, "Input (plain text)", pText));

System.out.println(String.format(OUTPUT\_FORMAT, "Key (hex)", CryptoUtils.hex(secretKey.getEncoded())));

System.out.println(String.format(OUTPUT\_FORMAT, "IV (hex)", CryptoUtils.hex(iv)));

System.out.println(String.format(OUTPUT\_FORMAT, "Encrypted (hex) ", CryptoUtils.hex(encryptedText)));

System.out.println(String.format(OUTPUT\_FORMAT, "Encrypted (hex) (block = 16)", CryptoUtils.hexWithBlockSize(encryptedText, 16)));

System.out.println("\n------ AES GCM Decryption ------");

System.out.println(String.format(OUTPUT\_FORMAT, "Input (hex)", CryptoUtils.hex(encryptedText)));

System.out.println(String.format(OUTPUT\_FORMAT, "Input (hex) (block = 16)", CryptoUtils.hexWithBlockSize(encryptedText, 16)));

System.out.println(String.format(OUTPUT\_FORMAT, "Key (hex)", CryptoUtils.hex(secretKey.getEncoded())));

String decryptedText = EncryptorAesGcm.decryptWithPrefixIV(encryptedText, secretKey);

System.out.println(String.format(OUTPUT\_FORMAT, "Decrypted (plain text)", decryptedText));

}

}

**CryptoUtils.java**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package com.sddpa.crypto;

/\*\*

\*

\* @author Ramu Maloth

\*/

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.SecretKeyFactory;

import javax.crypto.spec.PBEKeySpec;

import javax.crypto.spec.SecretKeySpec;

import java.security.NoSuchAlgorithmException;

import java.security.SecureRandom;

import java.security.spec.InvalidKeySpecException;

import java.security.spec.KeySpec;

import java.util.ArrayList;

import java.util.List;

public class CryptoUtils {

public static byte[] getRandomNonce(int numBytes) {

byte[] nonce = new byte[numBytes];

new SecureRandom().nextBytes(nonce);

return nonce;

}

// AES secret key

public static SecretKey getAESKey(int keysize) throws NoSuchAlgorithmException {

KeyGenerator keyGen = KeyGenerator.getInstance("AES");

keyGen.init(keysize, SecureRandom.getInstanceStrong());

SecretKey sk = keyGen.generateKey();

//System.out.println("Secret key is "+sk.getAlgorithm());

return keyGen.generateKey();

}

// Password derived AES 256 bits secret key

public static SecretKey getAESKeyFromPassword(char[] password, byte[] salt)

throws NoSuchAlgorithmException, InvalidKeySpecException {

SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");

// iterationCount = 65536

// keyLength = 256

KeySpec spec = new PBEKeySpec(password, salt, 65536, 256);

SecretKey secret = new SecretKeySpec(factory.generateSecret(spec).getEncoded(), "AES");

return secret;

}

// hex representation

public static String hex(byte[] bytes) {

StringBuilder result = new StringBuilder();

for (byte b : bytes) {

result.append(String.format("%02x", b));

}

return result.toString();

}

// print hex with block size split

public static String hexWithBlockSize(byte[] bytes, int blockSize) {

String hex = hex(bytes);

// one hex = 2 chars

blockSize = blockSize \* 2;

// better idea how to print this?

List<String> result = new ArrayList<>();

int index = 0;

while (index < hex.length()) {

result.add(hex.substring(index, Math.min(index + blockSize, hex.length())));

index += blockSize;

}

return result.toString();

}

}

SHA512.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package com.sddpa.crypto;

/\*\*

\*

\* @author Ramu Maloth

\*/

import com.google.common.io.BaseEncoding;

import com.google.common.primitives.Bytes;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.nio.charset.StandardCharsets;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.security.SecureRandom;

/\*\*

\* SHA512 hashing sample with plain Java. Uses a salt, configures the number of iterations and calculates the hash

\* value.

\* <p/>

\* Uses Google Guava to hex the hash in a readable format.

\*

\* @author Dominik Schadow

\*/

public class SHA512 {

private static final Logger log = LoggerFactory.getLogger(SHA512.class);

private static final String ALGORITHM = "SHA-512";

private static final int ITERATIONS = 1000000;

private static final int SALT\_SIZE = 64;

/\*\*

\* Private constructor.

\*/

private SHA512() {

}

public static void main(String[] args) {

String password = "TotallySecurePassword12345";

try {

byte[] salt = generateSalt();

log.info("Password {}. hash algorithm {}, iterations {}, salt {}", password, ALGORITHM, ITERATIONS,BaseEncoding.base16().encode(salt));

byte[] hash = calculateHash(password, salt);

boolean correct = verifyPassword(hash, password, salt);

System.out.println("Hash is "+hash);

log.info("Entered password is correct: {}", correct);

} catch (NoSuchAlgorithmException ex) {

log.error(ex.getMessage(), ex);

}

}

private static byte[] generateSalt() {

SecureRandom random = new SecureRandom();

byte[] salt = new byte[SALT\_SIZE];

random.nextBytes(salt);

return salt;

}

private static byte[] calculateHash(String password, byte[] salt) throws NoSuchAlgorithmException {

MessageDigest md = MessageDigest.getInstance(ALGORITHM);

md.reset();

md.update(Bytes.concat(password.getBytes(StandardCharsets.UTF\_8), salt));

byte[] hash = md.digest();

for (int i = 0; i < ITERATIONS; i++) {

md.reset();

hash = md.digest(hash);

}

return hash;

}

private static boolean verifyPassword(byte[] originalHash, String password, byte[] salt) throws

NoSuchAlgorithmException {

byte[] comparisonHash = calculateHash(password, salt);

log.info("hash 1: {}", BaseEncoding.base16().encode(originalHash));

log.info("hash 2: {}", BaseEncoding.base16().encode(comparisonHash));

return comparePasswords(originalHash, comparisonHash);

}

/\*\*

\* Compares the two byte arrays in length-constant time using XOR.

\*

\* @param originalHash The original password hash

\* @param comparisonHash The comparison password hash

\* @return True if both match, false otherwise

\*/

private static boolean comparePasswords(byte[] originalHash, byte[] comparisonHash) {

int diff = originalHash.length ^ comparisonHash.length;

for (int i = 0; i < originalHash.length && i < comparisonHash.length; i++) {

diff |= originalHash[i] ^ comparisonHash[i];

}

return diff == 0;

}

}

MD5Utility.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package com.sddpa.utils;

import com.sddpa.db.DBConnections;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.util.ArrayList;

import java.util.List;

/\*\*

\*

\* @author Ramu Maloth

\*/

public class Md5Auditutiliy {

private static PreparedStatement ps = null;

private static ResultSet rs = null;

public static List getOwnerDetails(int fileid) {

List list = new ArrayList();

try (Connection con = DBConnections.getDBConnection()){

String sqlQuery = "select ownername,email, publickey,filename from ownersfiles where id = ?";

ps = con.prepareStatement(sqlQuery);

ps.setInt(1, fileid);

rs = ps.executeQuery();

if(rs.next()){

list.add(rs.getString("ownername"));

list.add(rs.getString("email"));

list.add(rs.getString("publickey"));

list.add(rs.getString("filename"));

}

} catch (Exception e) {

e.printStackTrace();

}

return list;

}

}

StoreFileInCloud.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package com.sddpa.utils;

import com.cloud.SimpleFTPClient;

import java.io.ByteArrayInputStream;

import java.io.File;

import java.io.InputStream;

import java.io.UnsupportedEncodingException;

/\*\*

\*

\* @author Ramu Maloth

\*/

public class StoreFileInCloud {

static SimpleFTPClient client;

File file;

public static void storeInCloud(String cipher,String filename) {

try {

InputStream input = getInputStream(cipher, "UTF-8");

client = new SimpleFTPClient();

client.setHost("ftp.drivehq.com");

client.setUser("dpHyd");

client.setPassword("lx160cm");

client.setRemoteFile("dph/"+filename+".txt");

boolean log = client.connect();

client.uploadFile(input);

} catch (Exception e) {

e.printStackTrace();

}

}

public static InputStream getInputStream(String str, String encoding) throws UnsupportedEncodingException {

return new ByteArrayInputStream(str.getBytes(encoding));

}

}

RSAGenerator.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package com.sddpa.rsa;

/\*\*

\*

\* @author Ramu Maloth

\*/

import java.math.\*;

import java.security.SecureRandom;

/\*\*

\* RSA Generator class is used to calculate the private and public keys, based

\* on a specified bit length. Encrypt and Decrypt Methods also defined below.

\*

\*/

public class RSAGenerator {

private Key publickey; //Public Key

private Key privatekey; //Private key

private static final BigInteger ONE = BigInteger.ONE; //Value of 1 expressed as a BigInteger for ease of calculation

/\*\*

\* Object Constructor

\*

\* @param numbits Bit Length used to generate the various components

\*/

public RSAGenerator(int numbits) {

//Generate p and q

BigInteger p = BigInteger.probablePrime(numbits, new SecureRandom());

BigInteger q = BigInteger.probablePrime(numbits, new SecureRandom());

//Compute n - modulus

BigInteger n = p.multiply(q);

//Compute Euler's totient function, phiN

BigInteger p\_minus\_one = p.subtract(ONE);

BigInteger q\_minus\_one = q.subtract(ONE);

BigInteger phiN = p\_minus\_one.multiply(q\_minus\_one);

//Calculate public exponent

BigInteger e, d;

do {

e = BigInteger.probablePrime(numbits, new SecureRandom());

} while ((e.compareTo(ONE) == 1) && (e.compareTo(phiN) == -1) && (e.gcd(phiN).compareTo(ONE) != 0));

//Calculate private exponent

d = e.modInverse(phiN);

//Set Keys

publickey = new Key(e, n);

privatekey = new Key(d, n);

}

public Key getPublicKey(){

return publickey;

}

public Key getPrivateKey(){

return privatekey;

}

/\*\*

\* Method used to encrypt a message string

\*

\* @param msg Message string to be encrypted

\* @return BigInteger value of encrypted message

\*/

public BigInteger encrypt(String msg) {

return (new BigInteger(msg.getBytes())).modPow(publickey.getComponent(), publickey.getModulus());

}

/\*\*

\* Method used to decrypt a message

\*

\* @param encrypt\_msg Encrypted message as a BigInteger

\* @return BigInteger value of decrypted string

\*/

public BigInteger decrypt(BigInteger encrypt\_msg) {

return encrypt\_msg.modPow(privatekey.getComponent(), privatekey.getModulus());

}

/\*\*

\* Prints to screen Public and Private key components

\*/

@Override

public String toString() {

return "Public Key -> " + publickey.toString()

+ "\nPrivate Key -> " + privatekey.toString();

}

}

Key.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package com.sddpa.rsa;

/\*\*

\*

\* @author Ramu Maloth

\*/

import java.math.BigInteger;

public class Key {

private BigInteger component; //Component

private BigInteger modulus; //Modulus

/\*\*

\* Object Constructor

\* @param component Component of Key

\* @param modulus Modulus

\*/

Key(BigInteger component, BigInteger modulus){

this.component = component;

this.modulus = modulus;

}

/\*\*

\* Method used to return the Component of the Key

\* @return BigInteger value of key's Component

\*/

public BigInteger getComponent(){

return component;

}

/\*\*

\* Method used to return the Modulus

\* @return BigInteger value of Modulus

\*/

public BigInteger getModulus(){

return modulus;

}

/\*\*

\* Prints to screen Key Information

\*/

@Override

public String toString(){

return "Component: "+component+" / Modulus: "+modulus;

}

}

DataownerRgisterAction.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package com.sddpa.actions;

import com.sddpa.db.DBConnections;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.Connection;

import java.sql.PreparedStatement;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\*

\* @author Ramu Maloth

\*/

public class DORegisterActions extends HttpServlet {

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

String custname = request.getParameter("custname").trim();

String loginid = request.getParameter("loginid").trim();

String pswd = request.getParameter("pswd").trim();

String mobile = request.getParameter("mobile").trim();

String email = request.getParameter("email").trim();

String locality = request.getParameter("locality").trim();

String address = request.getParameter("address").trim();

String city = request.getParameter("city").trim();

String state = request.getParameter("state").trim();

String status = "do";

//Connection con = null;

PreparedStatement ps = null;

try(Connection con = DBConnections.getDBConnection()){

String sqlQuery = "insert into dataowners(custname,loginid,pswd,mobile,email,locality,address,city,state,status) values(?,?,?,?,?,?,?,?,?,?)";

ps = con.prepareStatement(sqlQuery);

ps.setString(1, custname);

ps.setString(2, loginid);

ps.setString(3, pswd);

ps.setString(4, mobile);

ps.setString(5, email);

ps.setString(6, locality);

ps.setString(7, address);

ps.setString(8, city);

ps.setString(9, state);

ps.setString(10, status);

int no = ps.executeUpdate();

if(no>0){

response.sendRedirect("DORegistrations.jsp?msg=success");

}else{

response.sendRedirect("DORegistrations.jsp?msg=failed");

}

}catch(Exception ex){

System.out.println("Exception at Registrations = "+ex.getMessage());

response.sendRedirect("DORegistrations.jsp?msg=failed");

}

}

/\*\*

\* Returns a short description of the servlet.

\*

\* @return a String containing servlet description

\*/

@Override

public String getServletInfo() {

return "Short description";

}// </editor-fold>

}

FileUpload.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package com.sddpa.actions;

import com.sddpa.crypto.CryptoUtils;

import com.sddpa.crypto.EncryptorAesGcm;

import com.sddpa.db.DBConnections;

import com.sddpa.rsa.RSAGenerator;

import com.sddpa.utils.CryptoUtilities;

import com.sddpa.utils.StoreFileInCloud;

import com.sddpa.wrapper.FileUtils;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.math.BigInteger;

import java.nio.charset.Charset;

import java.nio.charset.StandardCharsets;

import java.security.NoSuchAlgorithmException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.util.Base64;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.crypto.SecretKey;

import javax.servlet.ServletConfig;

import javax.servlet.ServletException;

import javax.servlet.annotation.MultipartConfig;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import javax.servlet.http.Part;

/\*\*

\*

\* @author Ramu Maloth

\*/

@MultipartConfig(maxFileSize = 100000l)

public class DOUploadAction extends HttpServlet {

private static int AES\_KEY\_BIT;

private static int IV\_LENGTH\_BYTE;

byte[] iv;

SecretKey secretKey = null;

private static Charset UTF\_8;

@Override

public void init(ServletConfig config) throws ServletException {

try {

super.init(config); //To change body of generated methods, choose Tools | Templates.

AES\_KEY\_BIT = 128;

IV\_LENGTH\_BYTE = 12;

iv = CryptoUtils.getRandomNonce(IV\_LENGTH\_BYTE);

//secretKey = CryptoUtils.getAESKey(AES\_KEY\_BIT);

UTF\_8 = StandardCharsets.UTF\_8;

} catch (Exception ex) {

Logger.getLogger(DOUploadAction.class.getName()).log(Level.SEVERE, null, ex);

}

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

Part part = request.getPart("file");

InputStream is = part.getInputStream();

HttpSession hs = request.getSession();

InputStreamReader isReader = new InputStreamReader(is);

//Creating a BufferedReader object

BufferedReader reader = new BufferedReader(isReader);

StringBuffer sb = new StringBuffer();

String str;

String data = null;

while ((str = reader.readLine()) != null) {

sb.append(str);

}

data = sb.toString();

String OUTPUT\_FORMAT = "%-30s:%s";

try {

secretKey = CryptoUtils.getAESKey(AES\_KEY\_BIT);

System.out.println("Secret Key " + secretKey);

String encodedKey = Base64.getEncoder().encodeToString(secretKey.getEncoded());

System.out.println("Secret Key is = " + encodedKey);

byte[] encryptedText = EncryptorAesGcm.encryptWithPrefixIV(data.getBytes(UTF\_8), secretKey, iv);

//System.out.println("\n------ AES GCM Encryption ------");

// System.out.println(String.format(OUTPUT\_FORMAT, "Input (plain text)", data));

System.out.println(String.format(OUTPUT\_FORMAT, "Key (hex)", CryptoUtils.hex(secretKey.getEncoded())));

//System.out.println(String.format(OUTPUT\_FORMAT, "IV (hex)", CryptoUtils.hex(iv)));

System.out.println(String.format(OUTPUT\_FORMAT, "Encrypted (hex) ", CryptoUtils.hex(encryptedText)));

//System.out.println(String.format(OUTPUT\_FORMAT, "Encrypted (hex) (block = 16)", CryptoUtils.hexWithBlockSize(encryptedText, 16)));

//StoreFileInCloud.storeInCloud(CryptoUtils.hex(encryptedText), CryptoUtils.hex(secretKey.getEncoded()));

// System.out.println("\n------ AES GCM Decryption ------");

//System.out.println(String.format(OUTPUT\_FORMAT, "Input (hex)", CryptoUtils.hex(encryptedText)));

// System.out.println(String.format(OUTPUT\_FORMAT, "Input (hex) (block = 16)", CryptoUtils.hexWithBlockSize(encryptedText, 16)));

System.out.println(String.format(OUTPUT\_FORMAT, "Key (hex)", CryptoUtils.hex(secretKey.getEncoded())));

String decryptedText = EncryptorAesGcm.decryptWithPrefixIV(encryptedText, secretKey);

System.out.println(String.format(OUTPUT\_FORMAT, "Decrypted (plain text)", decryptedText));

} catch (Exception e) {

e.printStackTrace();

}

PreparedStatement ps = null;

try(Connection con = DBConnections.getDBConnection()) {

RSAGenerator rsa = new RSAGenerator(64);

String c\_crypto\_data = CryptoUtilities.encrypt(data);

// Storing File in DriveHQ Cloud Server

//StoreFileInCloud.storeInCloud(CryptoUtils.hex(c\_crypto\_data.getBytes()), CryptoUtils.hex(secretKey.getEncoded()));

System.out.println("Data =" + c\_crypto\_data);

BigInteger private\_key = rsa.getPrivateKey().getComponent();

BigInteger public\_key = rsa.getPublicKey().getComponent();

String secretkey = Base64.getEncoder().encodeToString(secretKey.getEncoded());

String shaHexKey = CryptoUtils.hex(secretKey.getEncoded());

int data\_length = data.length();

String email = hs.getAttribute("email").toString();

String ownername = hs.getAttribute("username").toString();

String fileName = part.getSubmittedFileName();

String cs\_status = "waiting";

String sql\_query = "insert into ownersfiles(ownername,email,privatekey,publickey,secretkey,hexkey,filename,datalength,data,cs\_status) values(?,?,?,?,?,?,?,?,?,?)";

ps = con.prepareStatement(sql\_query);

ps.setString(1, ownername);

ps.setString(2, email);

ps.setLong(3, private\_key.longValue());

ps.setLong(4, public\_key.longValue());

ps.setString(5, secretkey);

ps.setString(6, shaHexKey);

ps.setString(7, fileName);

ps.setInt(8, data\_length);

ps.setString(9,c\_crypto\_data);

ps.setString(10,cs\_status);

ps.executeUpdate();

response.sendRedirect("DOUploadFile.jsp?msg=success");

//System.out.println("In The Server Secret Ket " + secretKey.toString());

//String cipher = FileUtils.encryptData(data, secretKey, iv);

//String plain = FileUtils.decryptData(cipher, secretKey);

// System.out.println("Cipher Blocks " + cipher);

// System.out.println("Plain Data " + plain);

} catch (Exception e) {

e.printStackTrace();

response.sendRedirect("DOUploadFile.jsp?msg=failed");

}

}

@Override

public String getServletInfo() {

return "Short description";

}// </editor-fold>

}