Tobias Hallen
16322468
09/04/2019

CS3031 Securing the Cloud

The program is divided into three separate segments, each of which run on their own. The three parts are initDB, Server and Client. InitDB is a class which initialises the RethinkDB database to contain the correct tables and indexes, i.e. a table for User data, 1 for File Data, and 1 which links Users to files, indicating authorised access, which is called FileKeys.

InitDB is a standalone program which must be run on its own before client or server in order for them to function correctly (Note that either way, the rethinkdb server must be running at all times, even before running initDB, as otherwise the db cannot be accessed or altered).
It is self-contained and very short, it simply inserts a database into the server, and fills that database with the tables:

Users - has indexes username (self explanatory) and a Public Key
Files - contains indexes name, owner (name is file name and owner is username of uploader of said file) and data (file data, encrypted with AES)
FileKeys - contains indexes name, owner and user (name is file name, owner is username of uploader of said file, user is the username of the user being granted access to the file in question by this filekey) along with a key value

Client is the second part of the program, and the one which the user interacts with. It features a simple GUI with an input line and an output field containing console output. Before inputting any other commands, a user must “register” using a username. A directory is then created in a “users” folder according to the name chosen, and in the new user-directory a PEM file containing a private key, using X.509 encoding is generated and stored. In future, commands made from this user will use the already-generated PEM file.
One can swap users from the GUI by inputting a “login” command, which switches to the specified user and their PEM file.
Registering will send a request to the server to insert a new entry into the users table. Upon successful registration, that user is now able to make any of the requests the program offers, namely:

- upload <filepath> <filename>
- download <user> <filename> <outputpath>
- share <filename> <user>...
- revoke <filename> <user>...

Upload takes the filepath of a file on disk and a name to give the uploaded file on server, and encrypts said file using a shared secret with AES256 encryption. Then, the file is signed with the user’s stored private key and is sent on to the server. The request is verified by the server before passing the file on to the database for insertion.
The generated shared key is then encrypted with the users own public key and uploaded into the filekey table, linking the owner to the file with the key.

Download takes the username of the owner, along with the filename on the database and a filepath to write the file to. It makes a GET request for the file, along with another GET request for the filekey linking them to the file. If the filekey exists, the shared key is decrypted using their private key, before the file is decrypted using the decrypted shared key. The file is then written to the specified directory. However, if there is no filekey entry linking this user to that file, the decryption will fail and no file will be written.

Share takes the filename and the user(s) with which to share the file. Users may only share files they directly own. The function queries the db for the filekey linking the owner to the file, and decodes the shared key used to decrypt the file.
For each user listed, the shared key is re-encrypted using that users public key, and uploaded as a new filekey entry along with the user's and file's names. When finished a new filekey entry will exist for each user listed, linking said user to the file.

Revoke takes a filename and the user(s) from whom to revoke file access. First, the user re-encrypts the file using AES encryption and re-uploads it to the database, overwriting the previous file entry. A new filekey is also uploaded tying the owner/reuploader to the re-uploaded file.
Then, for each user listed, a request is made to the /revoke endpoint, which deletes a filekey entry from the table.
However, since the file has been re-encrypted, file access to the file from people not being revoked must be updated. The newly re-uploaded file is re-shared with all users who previously had access to the file and did not have access revoked.
This way, users who have had their access revoked can no longer decrypt the file in question, where others who had access to it still can.

To facilitate all of theses functions there are other endpoints to get various files, users and filekeys from the tables in the database.

Any request made by the client is forwarded to one of several of the Server class's HTTP endpoints, which make requests to the rethink database server and return data in a JSON format.

The server class is quite simple in nature, and much of it was explained above. It is a simple routed HTTP server, which handles making requests to and receiving information from the database using queries. All data is handled in JSON format. The server verifies each POST request (except registering) being made by the signature the data in the request using the public key stored on the database. If the signature cannot be verified, the request is rejected and a negative HTTP response is returned.

The project is fairly easily adaptable to other cloud-based storage systems, though one would still require some form of hosted server to store public keys and the like.

Below is a listing of Open Source libraries/tools/dependencies used to create this project, and their individual purposes/functions.

The database is RethinkDB, a noSQL offshoot, used to store users, files and keys. A library is provided for interfacing using java.

The gson library is used for handling JSON objects, conversions to/from POJO's, etc.

The GUI for the client is implemented using the java Swing library.

The server class uses the Express-Java library, which is a very lightweight http/routing library for handling/sending http requests/responses.

File/KeyPair Cryptography was implemented using standard java security and crypto libraries, along with BouncyCastle to write to and read from PEM files.

- Encrypting shared secret keys and signings - RSA encryption
- Private key stored in X.509 encoded PEM file
- File Encryption using AES256 encryption

```java
package client;

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.io.PrintStream;
import java.nio.file.Files;
import java.security.GeneralSecurityException;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.spec.X509EncodedKeySpec;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.BoxLayout;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.UnsupportedLookAndFeelException;
import org.apache.commons.io.FileUtils;
import org.apache.http.client.ClientProtocolException;

import com.google.gson.Gson;

import serverUtil.TextAreaOutputStream;

public class Client
{
    final JFileChooser fc = new JFileChooser();
    public static PublicKey publicKey;
    public static PrivateKey privateKey;
    public static KeyPair kp;
    public static String ClientUser = "default";

    public static void main(String[] args) throws IOException, GeneralSecurityException,
  UnsupportedLookAndFeelException
    {

        initialize();
    }

    public static void Register(String userName) throws ClientProtocolException,
  IOException, GeneralSecurityException
    {
        KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
        kpg.initialize(1024);
        kp = ClientCrypto.generateKeyPair(userName);
        privateKey=kp.getPrivate();
        publicKey=kp.getPublic();
        serverUtil.User u = new serverUtil.User(userName, publicKey.getEncoded());
        ClientUserFunctions.Register(u);
```

```
61          System.out.println("Registered Successfully!");
62      }
63
64      public static void Upload(String filepath, String filename) throws IOException,
    GeneralSecurityException
65      {
66          java.io.File file = new java.io.File(filepath);
67          SecretKey sk = ClientCrypto.generateAESKey();
68          byte[] b = Files.readAllBytes(file.toPath());
69          byte[] encoded = ClientCrypto.encryptAES(sk, b);
70          ClientFile f = new ClientFile(ClientUser, filename, encoded);
71          ClientFileFunctions.Upload(f, privateKey);
72          byte[] encodedKey = ClientCrypto.encrypt(sk.getEncoded(), publicKey);
73          ClientFileKey fk = new ClientFileKey(ClientUser, ClientUser, filename, encodedKey);
74          ClientFileKeyFunctions.Share(fk, privateKey);
75          System.out.println("Successfully Uploaded File!");
76      }
77
78      public static void SwitchUsers(String newUser) throws IOException,
    GeneralSecurityException
79      {
80          java.io.File pem = new java.io.File("users/"+newUser+"/privateKey.pem");
81          if(pem.exists())
82          {
83              ClientUser=newUser;
84              kp = ClientCrypto.generateKeyPair(newUser);
85              privateKey=kp.getPrivate();
86              publicKey=kp.getPublic();
87              System.out.println("Logged in as: "+newUser);
88          }
89          else
90              System.out.println("User does not Exist, please register user before switching
    to it.");
91      }
92
93      public static void Download(String owner, String filename, String outPath) throws
    ClientProtocolException, IOException, GeneralSecurityException
94      {
95          ClientFile f = ClientFileFunctions.GetFile(owner, filename, ClientUser);
96          ClientFileKey fk = ClientFileKeyFunctions.GetFileKey(owner, filename, ClientUser);
97
98          byte[] decodedData = "".getBytes();
99          try {
100             byte[] decodedKey = ClientCrypto.decrypt(fk.key, privateKey);
101             decodedData = ClientCrypto.decryptAES(new SecretKeySpec(decodedKey, 0,
    decodedKey.length,"DES"), f.data);
102             FileUtils.writeByteArrayToFile(new java.io.File("C:\\Users\\artha\\Documents\
    \GitHub\\Cloud_Encryption\\Cloud_Encryption\\upANDdownloadFolder"+"\
    \noDecryption"+filename), f.data);
103             FileUtils.writeByteArrayToFile(new java.io.File(outPath), decodedData);
104             System.out.println("Successfully Downloaded File!");
105         } catch (javax.crypto.BadPaddingException e)
106         {
107             System.out.println("Error Decrypting: You do not have access to this File!");
108         }
109
110     }
111
112     public static void Share(String filename, String[] Users) throws
    ClientProtocolException, IOException, GeneralSecurityException
113     {
114         ClientFileKey fk = ClientFileKeyFunctions.GetFileKey(ClientUser, filename,
```

```
ClientUser);
115         byte[] decodedKey = ClientCrypto.decrypt(fk.key, privateKey);
116         for(String u : Users)
117         {
118             if(!u.equals(ClientUser))
119             {
120                 serverUtil.User user = ClientUserFunctions.GetUser(u);
121                 fk.id="";
122                 fk.user=u;
123                 byte[] encodedKey = ClientCrypto.encrypt(decodedKey,
KeyFactory.getInstance("RSA").generatePublic(new X509EncodedKeySpec(user.PubKey)));
124                 fk.key=encodedKey;
125                 ClientFileKeyFunctions.Share(fk, privateKey);
126             }
127         }
128         System.out.println("File Shared Successfully!");
129     }
130
131     public static void Revoke(String filename, String[] users) throws
ClientProtocolException, IOException, GeneralSecurityException
132     {
133         ClientFile f = ClientFileFunctions.GetFile(ClientUser, filename, ClientUser);
134         ClientFileKey fk = ClientFileKeyFunctions.GetFileKey(ClientUser, filename,
ClientUser);
135
136         byte[] decodedKey = ClientCrypto.decrypt(fk.key, privateKey);
137         byte[] decodedData = ClientCrypto.decryptAES(new SecretKeySpec(decodedKey, 0,
decodedKey.length,"DES"), f.data);
138         SecretKey sk = ClientCrypto.generateAESKey();
139         byte[] encodedData = ClientCrypto.encryptAES(sk, decodedData);
140         f.data=encodedData;
141
142         ClientFileFunctions.Upload(f, privateKey);
143
144         byte[] encodedKey=ClientCrypto.encrypt(sk.getEncoded(), publicKey);
145         fk.key=encodedKey;
146         new Gson();
147
148         ClientFileKeyFunctions.Share(fk, privateKey);
149
150         for(String u:users)
151         {
152             ClientFileKey fileKey = new ClientFileKey(u, ClientUser, filename, null);
153             ClientFileKeyFunctions.Revoke(fileKey, privateKey);
154         }
155         List<String> fUsers = ClientFileFunctions.GetFileUsers(ClientUser, filename,
ClientUser);
156         Gson gson = new Gson();
157         List<String> s = new ArrayList<String>();
158         for(Object o : fUsers)
159         {
160             FileUsers fu = gson.fromJson(gson.toJson(o), FileUsers.class);
161             if(!fu.user.equals(f.owner))
162             {
163                 s.add(fu.user);
164             }
165         }
166         Share(filename, s.toArray(new String[0]));
167 //      if(fUsers!=null)
168 //      {
169 //          Share(filename, fUsers.toString());
170 //      }
```

```
171         System.out.println("Successfully Revoked File!");
172     }
173 
174     private static void initialize()
175     {
176         final JFrame frame = new JFrame();
177         frame.add( new JLabel("Client" ), BorderLayout.NORTH );
178         JTextArea ta = new JTextArea(800, 400);
179         TextAreaOutputStream taos = new TextAreaOutputStream(ta);
180         PrintStream ps = new PrintStream(taos);
181         System.setOut(ps);
182         System.setErr(ps);
183 
184         final JTextField input = new JTextField("Input", 50);
185         input.requestFocus();
186         ActionListener a = new ActionListener()
187         {
188 
189             public void actionPerformed(ActionEvent e)
190             {
191                 String[] inputArray = input.getText().split("\\s+");
192                 try {
193                     if(inputArray[0].equals("register"))
194                     {
195                         input.setText("");
196                         ClientUser = inputArray[1];
197                         Register(ClientUser);
198                     }
199                     else if(inputArray[0].equals("login"))
200                     {
201                         input.setText("");
202                         SwitchUsers(inputArray[1]);
203                     }
204                     else if(inputArray[0].equals("upload"))
205                     {
206                         input.setText("");
207                         Upload(inputArray[1], inputArray[2]);
208                     }
209                     else if(inputArray[0].equals("download"))
210                     {
211                         input.setText("");
212                         Download(inputArray[1], inputArray[2], inputArray[3]);
213                     }
214                     else if(inputArray[0].equals("share"))
215                     {
216                         input.setText("");
217                         Share(inputArray[1], java.util.Arrays.copyOfRange(inputArray, 2,
    inputArray.length));
218                     }
219                     else if(inputArray[0].equals("revoke"))
220                     {
221                         input.setText("");
222                         Revoke(inputArray[1], java.util.Arrays.copyOfRange(inputArray, 2,
    inputArray.length));
223                     }
224                     else
225                     {
226                         input.setText("");
227                         System.out.println("Bad Input");
228                     }
229                 }
230                 catch (IOException io)
```

```
231                 {
232                     System.out.println("IOException in Input");
233                 } catch (GeneralSecurityException e1) {
234                     System.out.println("GeneralSecurityException in Input");
235                 }
236 
237             }};
238             input.addActionListener(a);
239             ta.setEditable(false);
240 
241             frame.getContentPane().setLayout(new BoxLayout(frame.getContentPane(),
    BoxLayout.Y_AXIS));
242             frame.getContentPane().add(input);
243             frame.getContentPane().add(new JScrollPane(ta));
244 
245             frame.pack();
246             frame.setVisible(true);
247             frame.setSize(800, 600);
248 
249             frame.addWindowListener(new java.awt.event.WindowAdapter() {
250                 @Override
251                 public void windowClosing(java.awt.event.WindowEvent windowEvent) {
252                     if (JOptionPane.showConfirmDialog(frame,
253                             "Are you sure you want to close the Client?", "Close Client?",
254                             JOptionPane.YES_NO_OPTION,
255                             JOptionPane.QUESTION_MESSAGE) == JOptionPane.YES_OPTION){
256                         System.exit(0);
257                     }
258                 }
259             });
260             frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
261             System.out.println("Client Ready for input!");
262 
263     }
264 
265 }
266 
```

```java
package client;

import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.security.GeneralSecurityException;
import java.security.InvalidKeyException;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.security.SignatureException;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import org.bouncycastle.openssl.PEMKeyPair;
import org.bouncycastle.openssl.PEMParser;
import org.bouncycastle.openssl.jcajce.JcaPEMKeyConverter;
import org.bouncycastle.openssl.jcajce.JcaPEMWriter;
import javax.crypto.*;
import org.apache.commons.codec.binary.Base64;

class ClientCrypto
{
    static KeyPair generateKeyPair(String clientName) throws IOException,
  GeneralSecurityException
    {

        new java.io.File("users/"+clientName).mkdirs();
        java.io.File pem = new java.io.File("users/"+clientName+"/privateKey.pem");
        if(!pem.exists())
        {

            KeyPairGenerator gen = KeyPairGenerator.getInstance("RSA");
            gen.initialize(2048);
            KeyPair kp = gen.generateKeyPair();
            PrintWriter pw = new PrintWriter(pem);
            JcaPEMWriter writer = new JcaPEMWriter(pw);
            writer.writeObject(kp.getPrivate());
            writer.writeObject(kp.getPublic());
            writer.close();

            return kp;
        }
        else
        {
            java.security.Security.addProvider(new
  org.bouncycastle.jce.provider.BouncyCastleProvider());
            PEMParser pemParser = new PEMParser(new FileReader(pem));
            JcaPEMKeyConverter converter = new JcaPEMKeyConverter().setProvider("BC");
            Object object = pemParser.readObject();
            KeyPair kp = converter.getKeyPair((PEMKeyPair) object);
            pemParser.close();
            return kp;
        }
    }

    static String sign(PrivateKey privateKey, String message) throws
  NoSuchAlgorithmException, InvalidKeyException, SignatureException,
  UnsupportedEncodingException {
```

```
59          Signature sign = Signature.getInstance("SHA1withRSA");
60          sign.initSign(privateKey);
61          sign.update(message.getBytes("UTF-8"));
62          return new String(Base64.encodeBase64(sign.sign()), "UTF-8");
63      }
64
65
66
67
68      static byte[] encrypt(byte[] rawText, PublicKey publicKey) throws IOException,
    GeneralSecurityException {
69          Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1PADDING");
70          cipher.init(Cipher.ENCRYPT_MODE,  (RSAPublicKey)publicKey);
71          return cipher.doFinal(rawText);
72      }
73
74      static byte[] decrypt(byte[] cipherText, PrivateKey privateKey) throws
    GeneralSecurityException {
75          Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1PADDING");
76          cipher.init(Cipher.DECRYPT_MODE, (RSAPrivateKey)privateKey);
77          return cipher.doFinal(cipherText);
78      }
79
80      static byte[] encryptAES(SecretKey key, byte[] value) throws NoSuchAlgorithmException,
    NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException
    {
81          Cipher c = Cipher.getInstance("AES/ECB/PKCS5PADDING");
82          c.init(Cipher.ENCRYPT_MODE, key);
83          byte[] cipherbytes = c.doFinal(value);
84          return cipherbytes;
85      }
86
87      static byte[] decryptAES(SecretKey key, byte[] encrypted) throws InvalidKeyException,
    IllegalBlockSizeException, BadPaddingException, NoSuchAlgorithmException,
    NoSuchPaddingException {
88          Cipher c = Cipher.getInstance("AES/ECB/PKCS5PADDING");
89          c.init(Cipher.DECRYPT_MODE, key);
90          byte[] decrypted = c.doFinal(encrypted);
91          return decrypted;
92      }
93
94      static SecretKey generateAESKey() throws NoSuchAlgorithmException
95      {
96          KeyGenerator keyGen = KeyGenerator.getInstance("AES");
97          keyGen.init(256); // for example
98          SecretKey secretKey = keyGen.generateKey();
99          return secretKey;
100     }
101 }
102
103
104
```

```java
package client;

class ClientFile
{
    public String id;
    public String owner;
    public String name;
    public byte[] data;
    ClientFile(String owner, String name, byte[] data) {
        super();
        this.owner = owner;
        this.name = name;
        this.data = data;
    }

}

```

```java
package client;

import java.io.IOException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.SignatureException;
import java.util.List;

import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.BasicResponseHandler;
import org.apache.http.impl.client.HttpClientBuilder;

import com.google.gson.Gson;

class ClientFileFunctions
{
    static void Upload(ClientFile f, PrivateKey pk) throws InvalidKeyException, NoSuchAlgorithmException, SignatureException, ClientProtocolException, IOException
    {
        Gson gson = new Gson();
        String message = gson.toJson(f);
        String signature = ClientCrypto.sign(pk, message);
        SignedRequest sr = new SignedRequest(message, signature);
//      System.out.println(signature);

        String signedJSON = gson.toJson(sr);
        HttpClient httpClient = HttpClientBuilder.create().build();
        HttpPost post = new HttpPost("http://80.111.202.166:8000/uploadfile");
        StringEntity postingString = new StringEntity(signedJSON);
        post.setEntity(postingString);
        post.setHeader("Content-type", "application/json");
        HttpResponse response = httpClient.execute(post);
        if(response.getStatusLine().getStatusCode()!=200)
        {
            System.out.println("Response was not Positive: "+response.getStatusLine().getStatusCode());
        }
        else System.out.println("Response was OK");
    }

    static ClientFile GetFile(String owner, String filename, String clientUser) throws InvalidKeyException, NoSuchAlgorithmException, SignatureException, ClientProtocolException, IOException
    {
        HttpClient httpClient = HttpClientBuilder.create().build();
        HttpGet get = new HttpGet("http://80.111.202.166:8000/"+"users/"+owner+"/"+filename);
        HttpResponse response = httpClient.execute(get);
        if(response.getStatusLine().getStatusCode()!=200)
        {
            System.out.println("Response was not Positive: "+response.getStatusLine().getStatusCode());
        }
        else System.out.println("Response was OK");
        String responseString = new BasicResponseHandler().handleResponse(response);
        Gson g = new Gson();
```

```
57          ClientFile f = g.fromJson(responseString, ClientFile.class);
58          return f;
59      }
60
61      static List GetFileUsers(String owner, String filename, String clientUser) throws
    InvalidKeyException, NoSuchAlgorithmException, SignatureException, ClientProtocolException,
    IOException
62      {
63          HttpClient httpClient = HttpClientBuilder.create().build();
64          HttpGet get = new
    HttpGet("http://80.111.202.166:8000/"+"users/"+owner+"/"+filename+"/users");
65          HttpResponse response = httpClient.execute(get);
66          if(response.getStatusLine().getStatusCode()!=200)
67          {
68              System.out.println("Response was not Positive:
    "+response.getStatusLine().getStatusCode());
69          }
70          else System.out.println("Response was OK");
71          String responseString = new BasicResponseHandler().handleResponse(response);
72          Gson g = new Gson();
73
74          return g.fromJson(responseString, List.class);
75
76      }
77
78  }
79
```

```java
package client;

class ClientFileKey
{
    public String id;
    public String user;
    public String owner;
    public String name;
    public byte[] key;

    ClientFileKey(String user, String owner, String name, byte[] key) {
        super();
        this.user = user;
        this.owner = owner;
        this.name = name;
        this.key = key;
    }
    ClientFileKey()
    {
        id = "";
        user = "";
        owner = "";
        name = "";
        key = "".getBytes();
    }

}

```

```java
package client;
import java.io.IOException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.SignatureException;

import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.BasicResponseHandler;
import org.apache.http.impl.client.HttpClientBuilder;

import com.google.gson.*;

class ClientFileKeyFunctions
{
    static void Share(ClientFileKey fk, PrivateKey pk) throws InvalidKeyException, NoSuchAlgorithmException, SignatureException, ClientProtocolException, IOException
    {
        Gson gson = new Gson();
        String message = gson.toJson(fk);
        String signature = ClientCrypto.sign(pk, message);
        SignedRequest sr = new SignedRequest(message, signature);
        String signedJSON = gson.toJson(sr);
        HttpClient httpClient = HttpClientBuilder.create().build();
        HttpPost post = new HttpPost("http://80.111.202.166:8000/sharefile");
        StringEntity postingString = new StringEntity(signedJSON);
        post.setEntity(postingString);
        post.setHeader("Content-type", "application/json");
        HttpResponse response = httpClient.execute(post);
        if(response.getStatusLine().getStatusCode()!=200)
        {
            System.out.println("Response was not Positive: "+response.getStatusLine().getStatusCode());
        }
        else System.out.println("Response was OK");
    }

    static void Revoke(ClientFileKey fk, PrivateKey pk) throws InvalidKeyException, NoSuchAlgorithmException, SignatureException, ClientProtocolException, IOException
    {
        Gson gson = new Gson();
        String message = gson.toJson(fk);
        String signature = ClientCrypto.sign(pk, message);
        SignedRequest sr = new SignedRequest(message, signature);
        String signedJSON = gson.toJson(sr);
        HttpClient httpClient = HttpClientBuilder.create().build();
        HttpPost post = new HttpPost("http://80.111.202.166:8000/revokefile");
        StringEntity postingString = new StringEntity(signedJSON);
        post.setEntity(postingString);
        post.setHeader("Content-type", "application/json");
        HttpResponse response = httpClient.execute(post);
        if(response.getStatusLine().getStatusCode()!=200)
        {
            System.out.println("Response was not Positive: "+response.getStatusLine().getStatusCode());
        }
        else System.out.println("Response was OK");
```

```
59      }
60
61      static ClientFileKey GetFileKey(String owner, String filename, String clientUser) throws
    InvalidKeyException, NoSuchAlgorithmException, SignatureException, ClientProtocolException,
    IOException
62      {
63          HttpClient httpClient = HttpClientBuilder.create().build();
64          HttpGet get = new
    HttpGet("http://80.111.202.166:8000/"+"users/"+owner+"/"+filename+"/key/"+clientUser);
65          HttpResponse response = httpClient.execute(get);
66          if(response.getStatusLine().getStatusCode()!=200)
67          {
68              System.out.println("Response was not Positive:
    "+response.getStatusLine().getStatusCode());
69              return new ClientFileKey();
70          }
71          else
72          {
73              System.out.println("Response was OK");
74              String responseString = new BasicResponseHandler().handleResponse(response);
75              Gson g = new Gson();
76              ClientFileKey fk= g.fromJson(responseString, ClientFileKey.class);
77              return fk;
78          }
79
80      }
81  }
82
```

ClientUserFunctions.java

```java
package client;
import java.io.IOException;
import com.google.gson.Gson;
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.util.EntityUtils;

class ClientUserFunctions
{
    static int Register(serverUtil.User u) throws ClientProtocolException, IOException
    {

        Gson gson = new Gson();
        HttpClient httpClient = HttpClientBuilder.create().build();
        HttpPost post = new HttpPost("http://80.111.202.166:8000/register");
        StringEntity postingString = new StringEntity(gson.toJson(u));
        post.setEntity(postingString);
        post.setHeader("Content-type", "application/json");
        HttpResponse response = httpClient.execute(post);
        if(response.getStatusLine().getStatusCode()!=200)
        {
            System.out.println("Response was not Positive: "+response.getStatusLine
().getStatusCode());
        }
        else System.out.println("Response was OK");
        return 0;
    }

    static serverUtil.User GetUser(String username) throws ClientProtocolException,
IOException
    {
        HttpClient httpClient = HttpClientBuilder.create().build();
        HttpGet get = new HttpGet("http://80.111.202.166:8000"+"/users/"+username);
        get.setHeader("Content-type", "application/json");
        HttpResponse response = httpClient.execute(get);
        String json = EntityUtils.toString(response.getEntity());
        Gson g = new Gson();
        return g.fromJson(json, serverUtil.User.class);

    }

}

```

```java
package client;

class FileUsers
{
    String user;

    FileUsers(String user) {
        super();
        this.user = user;
    }
}

```

```java
package client;

class SignedRequest
{
    public String message;
    public String signature;
    SignedRequest(String message, String signature) {
        super();
        this.message = message;
        this.signature = signature;
    }
}

```

```java
package initDB;

import org.apache.log4j.BasicConfigurator;

import com.rethinkdb.*;
import com.rethinkdb.net.Connection;


class InitDB
{
    public static final RethinkDB r = RethinkDB.r;

    public static void main(String[] args)
    {
        BasicConfigurator.configure();

        String DBHost = "127.0.0.1";
        Connection conn = r.connection().hostname(DBHost).port(28015).connect();

        r.dbCreate("Cloud_Encryption").run(conn);
        r.db("Cloud_Encryption").tableCreate("users").run(conn);
        r.db("Cloud_Encryption").table("users").indexCreate("username").run(conn);

        r.db("Cloud_Encryption").tableCreate("files").run(conn);
        r.db("Cloud_Encryption").table("files").indexCreate("name").run(conn);
        r.db("Cloud_Encryption").table("files").indexCreate("owner").run(conn);

        r.db("Cloud_Encryption").tableCreate("filekeys").run(conn);
        r.db("Cloud_Encryption").table("filekeys").indexCreate("name").run(conn);
        r.db("Cloud_Encryption").table("filekeys").indexCreate("owner").run(conn);
        r.db("Cloud_Encryption").table("filekeys").indexCreate("user").run(conn);

        System.exit(0);
    }
}


```

```java
package server;

import java.util.HashMap;

import com.rethinkdb.RethinkDB;
import com.rethinkdb.gen.ast.Table;
import com.rethinkdb.net.Connection;
import com.rethinkdb.net.Cursor;

public class File
{
    private static String DBHost = "127.0.0.1";
    private static final RethinkDB r = RethinkDB.r;
    private static Connection conn = r.connection().hostname(DBHost).port(28015).connect();
    private static Table filetable = r.db("Cloud_Encryption").table("files");


    static int insert(serverUtil.File f)
    {
        if(filetable.g("name").contains(f.name).run(conn))
        {
            if(filetable.g("owner").contains(f.owner).run(conn))
            {
                    System.out.println("Duplicate File Entry");
                    serverUtil.File file = getFile(f.owner, f.name);
                    f.id = file.id;
                    filetable.insert(r.hashMap("name", f.name).with("owner", f.owner).with("id", f.id).with("data", r.binary(f.data))).optArg("conflict", "replace").run(conn);
                    return 1;

            }
        }
        filetable.insert(r.hashMap("name", f.name).with("owner", f.owner).with("data", r.binary(f.data))).run(conn);
        return 0;
    }



    @SuppressWarnings("rawtypes")
    static serverUtil.File getFile(String Owner, String filename)
    {
        try {
            Cursor dbRes = filetable.getAll(filename).optArg("index", "name").filter(r.hashMap("owner", Owner)).run(conn);
            HashMap m = (HashMap) dbRes.next();
            serverUtil.File f = new serverUtil.File((String)m.get("id"), (String)m.get("owner"), (String)m.get("name"), (byte[])m.get("data"));
            return f;
        } catch (java.util.NoSuchElementException e) {
            System.out.println("Invalid file access");
            System.out.println("File "+filename+" owned by "+Owner+" does not exist, perhaps unauthorized access!");
        }
        return new serverUtil.File();
    }

}

```

```java
package server;
import com.google.gson.Gson;
import com.rethinkdb.*;
import com.rethinkdb.gen.ast.Table;
import com.rethinkdb.net.Connection;
import com.rethinkdb.net.Cursor;

import java.util.HashMap;
import java.util.List;

public class FileKey
{
    private static String DBHost = "127.0.0.1";
    private static final RethinkDB r = RethinkDB.r;
    private static Connection conn = r.connection().hostname(DBHost).port(28015).connect();
    private static Table filekeytable = r.db("Cloud_Encryption").table("filekeys");


    static int insert(serverUtil.FileKey fk)
    {
        if(filekeytable.g("name").contains(fk.name).run(conn))
        {
            if(filekeytable.g("owner").contains(fk.owner).run(conn))
            {
                if(filekeytable.g("user").contains(fk.user).run(conn))
                {
                    System.out.println("Duplicate FileKey Entry");
                    serverUtil.FileKey f = getFileKey(fk.owner, fk.name, fk.user);
                    fk.id = f.id;
                    filekeytable.insert(r.hashMap("name", fk.name).with("id", fk.id).with("user", fk.user).with("owner", fk.owner).with("key", r.binary(fk.key))).optArg("conflict", "replace").run(conn);
                    return 1;
                }
            }
        }
        filekeytable.insert(r.hashMap("name", fk.name).with("user", fk.user).with("owner", fk.owner).with("key", r.binary(fk.key))).run(conn);
        return 0;
    }



    static int revoke(serverUtil.FileKey fk)
    {
        if(fk.user == fk.owner)
        {
            System.out.println("Cannot revoke own file access");
            return 2;
        }
        if(filekeytable.g("name").contains(fk.name).run(conn))
        {
            if(filekeytable.g("owner").contains(fk.owner).run(conn))
            {
                if(filekeytable.g("user").contains(fk.user).run(conn))
                {

                    filekeytable.get(getFileKey(fk.owner, fk.name, fk.user).id).delete().run(conn);
                    return 0;
                }
            }
```

```
59          }
60          return 1;
61      }
62
63      @SuppressWarnings("rawtypes")
64      static serverUtil.FileKey getFileKey(String owner, String filename, String user)
65      {
66
67          Cursor dbRes;
68          try {
69              dbRes = filekeytable.getAll(filename).optArg("index",
  "name").filter(r.hashMap("owner", owner).with("user", user)).run(conn);
70              HashMap m = (HashMap) dbRes.next();
71              serverUtil.FileKey fk = new serverUtil.FileKey((String)m.get("id"),
  (String)m.get("user"), (String)m.get("owner"), (String)m.get("name"), (byte[])m.get("key"));
72              return fk;
73          } catch (java.util.NoSuchElementException e) {
74              System.out.println("Invalid file access");
75              System.out.println("User "+user+" does not have access to file "+filename+"
  owned by "+owner);
76          }
77          return new serverUtil.FileKey();
78      }
79
80  @SuppressWarnings("rawtypes")
81  static List getFileUsers(String owner, String filename)
82  {
83
84      Cursor dbRes = filekeytable.getAll(filename).optArg("index",
  "name").filter(r.hashMap("owner", owner)).pluck("user").run(conn);
85  //  System.out.println(dbRes.toList());
86  //  Gson gson = new Gson();
87  //  System.out.println( gson.toJson(dbRes.toList()));
88      return dbRes.toList();
89  }
90
91  }
92
```

```java
package server;
import java.awt.BorderLayout;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

import java.io.IOException;
import java.io.PrintStream;

import express.Express;
import serverUtil.TextAreaOutputStream;

public class Server
{
    public static void main(String[] args) throws IOException
    {
        final JFrame frame = new JFrame();
        frame.add( new JLabel("Server" ), BorderLayout.NORTH );
        JTextArea ta = new JTextArea();
        TextAreaOutputStream taos = new TextAreaOutputStream(ta);
        PrintStream ps = new PrintStream(taos);
        System.setOut(ps);
        System.setErr(ps);
        ta.setEditable(false);
        frame.add(new JScrollPane(ta));
        frame.pack();
        frame.setVisible(true);
        frame.setSize(800, 600);

        frame.addWindowListener(new java.awt.event.WindowAdapter() {
            @Override
            public void windowClosing(java.awt.event.WindowEvent windowEvent) {
                if (JOptionPane.showConfirmDialog(frame,
                    "Are you sure you want to close the Server?", "Close Server?",
                    JOptionPane.YES_NO_OPTION,
                    JOptionPane.QUESTION_MESSAGE) == JOptionPane.YES_OPTION){
                    System.exit(0);
                }
            }
        });
        frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);

        Express app = new Express();
        try {
            System.out.println("Starting up Server...");
            app.bind(new ServerBindings()); // See class below
            app.listen(8000);
        } catch (java.lang.ExceptionInInitializerError e) {
            System.out.println("Server Refused Connection. Shutting Down.");
            System.exit(0);
        }
    }

}

```

```java
package server;

import java.io.IOException;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.SignatureException;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.X509EncodedKeySpec;
import java.util.List;

import com.google.gson.Gson;
import com.google.gson.JsonIOException;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import com.google.gson.JsonSyntaxException;
import express.DynExpress;
import express.http.RequestMethod;
import express.http.request.Request;
import express.http.response.Response;
import express.utils.Status;

class ServerBindings {

    @DynExpress(context= "/register", method = RequestMethod.POST) // Default is context="/" and method=RequestMethod.GET
    public void register(Request req, Response res) throws IOException {
        System.out.println("Received Register Request!");
        JsonParser jsonParser = new JsonParser();
        JsonObject jsonObject = (JsonObject)jsonParser.parse(
                new InputStreamReader(req.getBody(), "UTF-8"));
        Gson gson = new Gson();
        String json = gson.toJson(jsonObject);
        serverUtil.User user = gson.fromJson(json,serverUtil.User.class);
        if(User.insert(user)!=0)res.send("Duplicate User Entry!");
        else res.send("User Registered!");

    }

    @DynExpress(context= "/uploadfile", method = RequestMethod.POST) // Only context is defined, method=RequestMethod.GET is used as method
    public void uploadFile(Request req, Response res) throws JsonIOException, JsonSyntaxException, UnsupportedEncodingException, InvalidKeyException, SignatureException, NoSuchAlgorithmException, InvalidKeySpecException {
        System.out.println("Received Upload Request!");
        JsonParser jsonParser = new JsonParser();
        JsonObject jsonObject = (JsonObject)jsonParser.parse(
                new InputStreamReader(req.getBody(), "UTF-8"));
        Gson gson = new Gson();
        String message = jsonObject.get("message").getAsString();
        String sig = jsonObject.get("signature").getAsString();

        serverUtil.File file = gson.fromJson(message, serverUtil.File.class);
        serverUtil.User u = User.GetUser(file.owner);
        if(!serverUtil.Crypto.verify(KeyFactory.getInstance("RSA").generatePublic(new X509EncodedKeySpec(u.PubKey)), message.getBytes(), sig))
        {
            System.out.println("Could not Verify Signature!");
            res.setStatus(Status.valueOf(403));
            res.send("Could not Verify Signature!");
```

```
58          }
59
60          if(File.insert(file)!=0) res.send("Updated Existing File!");
61          else res.send("File Uploaded!");
62      }
63
64      @DynExpress(context = "/sharefile", method = RequestMethod.POST) // Both defined
65      public void shareFile(Request req, Response res) throws JsonIOException,
    JsonSyntaxException, UnsupportedEncodingException, InvalidKeyException, SignatureException,
    NoSuchAlgorithmException, InvalidKeySpecException {
66          System.out.println("Received Share Request!");
67          JsonParser jsonParser = new JsonParser();
68          JsonObject jsonObject = (JsonObject)jsonParser.parse(
69                  new InputStreamReader(req.getBody(), "UTF-8"));
70          Gson gson = new Gson();
71          String message = jsonObject.get("message").getAsString();
72          String sig = jsonObject.get("signature").getAsString();
73
74          serverUtil.FileKey fk = gson.fromJson(message,serverUtil.FileKey.class);
75          serverUtil.User u = User.GetUser(fk.owner);
76          if(!serverUtil.Crypto.verify(KeyFactory.getInstance("RSA").generatePublic(new
    X509EncodedKeySpec(u.PubKey)), message.getBytes(), sig))
77          {
78              System.out.println("Could not Verify Signature!");
79              res.setStatus(Status.valueOf(403));
80              res.send("Could not Verify Signature!");
81          }
82
83          if(FileKey.insert(fk)!=0) res.send("Updated Existing FileKey!");
84          else res.send("File Shared!");
85      }
86
87      @DynExpress(context= "/revokefile", method = RequestMethod.POST) // Only the method is
    defined, "/" is used as context
88      public void revokeFile(Request req, Response res) throws JsonIOException,
    JsonSyntaxException, UnsupportedEncodingException, InvalidKeyException, SignatureException,
    NoSuchAlgorithmException, InvalidKeySpecException {
89          System.out.println("Received Revoke Request!");
90          JsonParser jsonParser = new JsonParser();
91          JsonObject jsonObject = (JsonObject)jsonParser.parse(
92                  new InputStreamReader(req.getBody(), "UTF-8"));
93          Gson gson = new Gson();
94          String message = jsonObject.get("message").getAsString();
95          String sig = jsonObject.get("signature").getAsString();
96
97          serverUtil.FileKey f1 = gson.fromJson(message, serverUtil.FileKey.class);
98          serverUtil.FileKey fk = FileKey.getFileKey(f1.owner, f1.name, f1.user);
99          if(!fk.id.equals(""))
100         {
101             serverUtil.User u = User.GetUser(fk.owner);
102             if(!serverUtil.Crypto.verify(KeyFactory.getInstance("RSA").generatePublic(new
    X509EncodedKeySpec(u.PubKey)), message.getBytes(), sig))
103             {
104                 System.out.println("Could not Verify Signature!");
105                 res.setStatus(Status.valueOf(403));
106                 res.send("Could not Verify Signature!");
107             }
108             else if(FileKey.revoke(fk)==2)
109             {
110                 res.send("Cannot revoke own file access!");
111             }
112             else if(FileKey.revoke(fk)==1)
```

```
113             {
114                 res.send("FileKey entry does not exist, cannot be revoked!");
115             }
116             else res.send("File access Revoked!");
117         }
118         else
119         {
120             res.send("FileKey entry does not exist, failed to revoke file access!");
121         }
122     }
123 
124     @DynExpress(context = "/users/:username", method = RequestMethod.GET) // Both defined
125     public void getUser(Request req, Response res) throws JsonIOException,
    JsonSyntaxException, UnsupportedEncodingException {
126         serverUtil.User u = User.GetUser(req.getParam("username"));
127         Gson gson = new Gson();
128         String json = gson.toJson(u);
129         res.send(json);
130     }
131 
132     @DynExpress(context = "/users/:username/:filename", method = RequestMethod.GET) // Both
    defined
133     public void getFile(Request req, Response res) {
134         serverUtil.File f = File.getFile(req.getParam("username"), req.getParam
    ("filename"));
135         if(f.name.equals(""))
136         {
137             res.setStatus(Status.valueOf(403));
138             res.send("Failed Getting File, does not Exist.");
139         }
140         else
141         {
142             Gson gson = new Gson();
143             String json = gson.toJson(f);
144             res.send(json);
145         }
146     }
147 
148     @SuppressWarnings("rawtypes")
149     @DynExpress(context = "/users/:username/:filename/users", method = RequestMethod.GET)
    // Both defined
150     public void getFileUsers(Request req, Response res) {
151         List l = FileKey.getFileUsers(req.getParam("username"), req.getParam("filename"));
152         Gson gson = new Gson();
153         String json = gson.toJson(l);
154         res.send(json);
155     }
156 
157     @DynExpress(context = "/users/:username/:filename/key/:user", method =
    RequestMethod.GET) // Both defined
158     public void getFileKey(Request req, Response res) {
159         serverUtil.FileKey fk = FileKey.getFileKey(req.getParam("username"), req.getParam
    ("filename"), req.getParam("user"));
160         if (fk.name.equals(""))
161         {
162             res.setStatus(Status.valueOf(403));
163             res.send("Failed Getting FileKey, unauthorized access.");
164         }
165         else
166         {
167             Gson gson = new Gson();
168             String json = gson.toJson(fk);
```

```
169             res.send(json);
170         }
171     }
172
173 }
174
```

```java
package server;

import com.google.gson.Gson;
import com.rethinkdb.*;
import com.rethinkdb.gen.ast.Table;
import com.rethinkdb.net.Connection;
import com.rethinkdb.net.Cursor;

public class User
{
    private static String DBHost = "127.0.0.1";
    private static final RethinkDB r = RethinkDB.r;
    private static Connection conn = r.connection().hostname(DBHost).port(28015).connect();
    private static Table userTable = r.db("Cloud_Encryption").table("users");

    public static int insert(serverUtil.User u)
    {
        if(userTable.g("username").contains(u.username).run(conn))
        {
            System.out.println("Duplicate User Entry");
            return 1;
        }
        else
        {
            serverUtil.DBUser user = new serverUtil.DBUser(u.id,u.username,u.PubKey.toString
());
            userTable.insert(r.hashMap("username", user.username).with("PubKey", r.binary
(u.PubKey))).run(conn);
            return 0;
        }
    }

    @SuppressWarnings("rawtypes")
    static serverUtil.User GetUser(String username)
    {
        if(userTable.g("username").contains(username).run(conn))
        {
            Cursor dbRes = userTable.getAll(username).optArg("index", "username").run(conn);
            for(Object doc : dbRes)
            {
                Gson gson = new Gson();
                String s = gson.toJson(doc);
                serverUtil.User u = gson.fromJson(s, serverUtil.User.class);
                return u;
            }
        }
        else System.out.println("User does not Exist");
        return null;
    }
}

```

```java
package serverUtil;
import java.security.*;
import java.util.Base64;


public class Crypto
{
    public static boolean verify(PublicKey pk, byte[] message, String signature) throws SignatureException, InvalidKeyException, NoSuchAlgorithmException
    {
        Signature publicSignature = Signature.getInstance("SHA1withRSA");
        publicSignature.initVerify(pk);
        publicSignature.update(message);

        byte[] signatureBytes = Base64.getDecoder().decode(signature);

        return publicSignature.verify(signatureBytes);
    }
}

```

```java
package serverUtil;

public class DBUser
{
    public DBUser(String iD, String username, String pubKey) {
        super();
        ID = iD;
        this.username = username;
        PubKey = pubKey;
    }
    public String ID = "gorethink:\"id,omitempty\"";
    public String username = "gorethink:\"username\"";
    public String PubKey = "gorethink:\"pubkey\"";
}

```

```java
package serverUtil;

public class File
{
    public String id;
    public String owner;
    public String name;
    public byte[] data;
    public File(String iD, String owner, String name, byte[] data) {
        super();
        id = iD;
        this.owner = owner;
        this.name = name;
        this.data = data;
    }
    public File()
    {
        id = "";
        owner = "";
        name = "";
        data = "".getBytes();
    }


}

```

```java
package serverUtil;

public class FileKey
{
    public String id;
    public String user;
    public String owner;
    public String name;
    public byte[] key;

    public FileKey(String iD, String user, String owner, String name, byte[] key) {
        super();
        id = iD;
        this.user = user;
        this.owner = owner;
        this.name = name;
        this.key = key;
    }



    public FileKey()
    {
        id = "";
        user = "";
        owner = "";
        name = "";
        key = "".getBytes();
    }

}

```

TextAreaOutputStream.java

```java
package serverUtil;

import java.io.IOException;
import java.io.OutputStream;
import javax.swing.JTextArea;

public class TextAreaOutputStream extends OutputStream {
    private JTextArea textControl;
    public TextAreaOutputStream( JTextArea control ) {
        textControl = control;
    }
    public void write( int b ) throws IOException {
        textControl.append( String.valueOf( ( char )b ) );
    }
}

```

```java
package serverUtil;

public class User
{
    public String id;
    public String username;
    public byte[] PubKey;

    public User(String username, byte[] pubKey) {
        super();
        this.username = username;
        PubKey = pubKey;
    }
}

```