

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 these 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

Client.java

```
1 package client;
2
3 import java.awt.BorderLayout;
4 import java.awt.event.ActionEvent;
5 import java.awt.event.ActionListener;
6 import java.io.IOException;
7 import java.io.PrintStream;
8 import java.nio.file.Files;
9 import java.security.GeneralSecurityException;
10 import java.security.KeyFactory;
11 import java.security.KeyPair;
12 import java.security.KeyPairGenerator;
13 import java.security.PrivateKey;
14 import java.security.PublicKey;
15 import java.security.spec.X509EncodedKeySpec;
16 import java.util.ArrayList;
17 import java.util.Arrays;
18 import java.util.List;
19
20 import javax.crypto.SecretKey;
21 import javax.crypto.spec.SecretKeySpec;
22 import javax.swing.BoxLayout;
23 import javax.swing.JFileChooser;
24 import javax.swing.JFrame;
25 import javax.swing.JLabel;
26 import javax.swing.JOptionPane;
27 import javax.swing.JScrollPane;
28 import javax.swing.JTextArea;
29 import javax.swing.JTextField;
30 import javax.swing.UnsupportedLookAndFeelException;
31 import org.apache.commons.io.FileUtils;
32 import org.apache.http.client.ClientProtocolException;
33
34 import com.google.gson.Gson;
35
36 import serverUtil.TextAreaOutputStream;
37
38 public class Client
39 {
40     final JFileChooser fc = new JFileChooser();
41     public static PublicKey publicKey;
42     public static PrivateKey privateKey;
43     public static KeyPair kp;
44     public static String ClientUser = "default";
45
46     public static void main(String[] args) throws IOException, GeneralSecurityException,
47         UnsupportedLookAndFeelException
48     {
49         initialize();
50     }
51
52     public static void Register(String userName) throws ClientProtocolException,
53         IOException, GeneralSecurityException
54     {
55         KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
56         kpg.initialize(1024);
57         kp = ClientCrypto.generateKeyPair(userName);
58         privateKey=kp.getPrivate();
59         publicKey=kp.getPublic();
60         serverUtil.User u = new serverUtil.User(userName, publicKey.getEncoded());
61         ClientUserFunctions.Register(u);
62     }
63 }
```

Client.java

```

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 outputPath) 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 = f.data.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(outputPath), 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,

```

Client.java

```

    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.GetFilesUsers(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 // }

```

Client.java

```

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)

```

Client.java

```

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 BorderLayout(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

```


ClientCrypto.java

```
1 package client;
2
3 import java.io.FileReader;
4 import java.io.IOException;
5 import java.io.PrintWriter;
6 import java.io.UnsupportedEncodingException;
7 import java.security.GeneralSecurityException;
8 import java.security.InvalidKeyException;
9 import java.security.KeyPair;
10 import java.security.KeyPairGenerator;
11 import java.security.NoSuchAlgorithmException;
12 import java.security.PrivateKey;
13 import java.security.PublicKey;
14 import java.security.Signature;
15 import java.security.SignatureException;
16 import java.security.interfaces.RSAPrivateKey;
17 import java.security.interfaces.RSAPublicKey;
18 import org.bouncycastle.openssl.PEMKeyPair;
19 import org.bouncycastle.openssl.PEMParser;
20 import org.bouncycastle.openssl.jcajce.JcaPEMKeyConverter;
21 import org.bouncycastle.openssl.jcajce.JcaPEMWriter;
22 import javax.crypto.*;
23 import org.apache.commons.codec.binary.Base64;
24
25 class ClientCrypto
26 {
27     static KeyPair generateKeyPair(String clientName) throws IOException,
        GeneralSecurityException
28     {
29
30         new java.io.File("users/"+clientName).mkdirs();
31         java.io.File pem = new java.io.File("users/"+clientName+"/privateKey.pem");
32         if(!pem.exists())
33         {
34
35             KeyPairGenerator gen = KeyPairGenerator.getInstance("RSA");
36             gen.initialize(2048);
37             KeyPair kp = gen.generateKeyPair();
38             PrintWriter pw = new PrintWriter(pem);
39             JcaPEMWriter writer = new JcaPEMWriter(pw);
40             writer.writeObject(kp.getPrivate());
41             writer.writeObject(kp.getPublic());
42             writer.close();
43
44             return kp;
45         }
46         else
47         {
48             java.security.Security.addProvider(new
        org.bouncycastle.jce.provider.BouncyCastleProvider());
49             PEMParser pemParser = new PEMParser(new FileReader(pem));
50             JcaPEMKeyConverter converter = new JcaPEMKeyConverter().setProvider("BC");
51             Object object = pemParser.readObject();
52             KeyPair kp = converter.getKeyPair((PEMKeyPair) object);
53             pemParser.close();
54             return kp;
55         }
56     }
57
58     static String sign(PrivateKey privateKey, String message) throws
        NoSuchAlgorithmException, InvalidKeyException, SignatureException,
        UnsupportedEncodingException {
```

ClientCrypto.java

```

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

```

ClientFile.java

```
1 package client;
2
3 class ClientFile
4 {
5     public String id;
6     public String owner;
7     public String name;
8     public byte[] data;
9     ClientFile(String owner, String name, byte[] data) {
10         super();
11         this.owner = owner;
12         this.name = name;
13         this.data = data;
14     }
15
16 }
17
```

ClientFileFunctions.java

```

1 package client;
2
3 import java.io.IOException;
4 import java.security.InvalidKeyException;
5 import java.security.NoSuchAlgorithmException;
6 import java.security.PrivateKey;
7 import java.security.SignatureException;
8 import java.util.List;
9
10 import org.apache.http.HttpResponse;
11 import org.apache.http.client.ClientProtocolException;
12 import org.apache.http.client.HttpClient;
13 import org.apache.http.client.methods.HttpGet;
14 import org.apache.http.client.methods.HttpPost;
15 import org.apache.http.entity.StringEntity;
16 import org.apache.http.impl.client.BasicResponseHandler;
17 import org.apache.http.impl.client.HttpClientBuilder;
18
19 import com.google.gson.Gson;
20
21 class ClientFileFunctions
22 {
23     static void Upload(ClientFile f, PrivateKey pk) throws InvalidKeyException,
        NoSuchAlgorithmException, SignatureException, ClientProtocolException, IOException
24     {
25         Gson gson = new Gson();
26         String message = gson.toJson(f);
27         String signature = ClientCrypto.sign(pk, message);
28         SignedRequest sr = new SignedRequest(message, signature);
29 //         System.out.println(signature);
30
31         String signedJSON = gson.toJson(sr);
32         HttpClient httpClient = HttpClientBuilder.create().build();
33         HttpPost post = new HttpPost("http://80.111.202.166:8000/uploadfile");
34         StringEntity postingString = new StringEntity(signedJSON);
35         post.setEntity(postingString);
36         post.setHeader("Content-type", "application/json");
37         HttpResponse response = httpClient.execute(post);
38         if(response.getStatusLine().getStatusCode()!=200)
39         {
40             System.out.println("Response was not Positive:
41 "+response.getStatusLine().getStatusCode());
42         }
43         else System.out.println("Response was OK");
44     }
45
46     static ClientFile GetFile(String owner, String filename, String clientUser) throws
        InvalidKeyException, NoSuchAlgorithmException, SignatureException, ClientProtocolException,
        IOException
47     {
48         HttpClient httpClient = HttpClientBuilder.create().build();
49         HttpGet get = new
50         HttpGet("http://80.111.202.166:8000/"+owner+"/"+filename);
51         HttpResponse response = httpClient.execute(get);
52         if(response.getStatusLine().getStatusCode()!=200)
53         {
54             System.out.println("Response was not Positive:
55 "+response.getStatusLine().getStatusCode());
56         }
57         else System.out.println("Response was OK");
58         String responseString = new BasicResponseHandler().handleResponse(response);
59         Gson g = new Gson();

```

ClientFileFunctions.java

```
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
```

ClientFileKey.java

```
1 package client;
2
3 class ClientFileKey
4 {
5     public String id;
6     public String user;
7     public String owner;
8     public String name;
9     public byte[] key;
10
11     ClientFileKey(String user, String owner, String name, byte[] key) {
12         super();
13         this.user = user;
14         this.owner = owner;
15         this.name = name;
16         this.key = key;
17     }
18     ClientFileKey()
19     {
20         id = "";
21         user = "";
22         owner = "";
23         name = "";
24         key = "".getBytes();
25     }
26
27 }
28
```

ClientFileKeyFunctions.java

```
1 package client;
2 import java.io.IOException;
3 import java.security.InvalidKeyException;
4 import java.security.NoSuchAlgorithmException;
5 import java.security.PrivateKey;
6 import java.security.SignatureException;
7
8 import org.apache.http.HttpResponse;
9 import org.apache.http.client.ClientProtocolException;
10 import org.apache.http.client.HttpClient;
11 import org.apache.http.client.methods.HttpGet;
12 import org.apache.http.client.methods.HttpPost;
13 import org.apache.http.entity.StringEntity;
14 import org.apache.http.impl.client.BasicResponseHandler;
15 import org.apache.http.impl.client.HttpClientBuilder;
16
17 import com.google.gson.*;
18
19 class ClientFileKeyFunctions
20 {
21     static void Share(ClientFileKey fk, PrivateKey pk) throws InvalidKeyException,
        NoSuchAlgorithmException, SignatureException, ClientProtocolException, IOException
22     {
23         Gson gson = new Gson();
24         String message = gson.toJson(fk);
25         String signature = ClientCrypto.sign(pk, message);
26         SignedRequest sr = new SignedRequest(message, signature);
27         String signedJSON = gson.toJson(sr);
28         HttpClient httpClient = HttpClientBuilder.create().build();
29         HttpPost post = new HttpPost("http://80.111.202.166:8000/sharefile");
30         StringEntity postingString = new StringEntity(signedJSON);
31         post.setEntity(postingString);
32         post.setHeader("Content-type", "application/json");
33         HttpResponse response = httpClient.execute(post);
34         if(response.getStatusLine().getStatusCode() != 200)
35         {
36             System.out.println("Response was not Positive:
37 "+response.getStatusLine().getStatusCode());
38         }
39         else System.out.println("Response was OK");
40     }
41
42     static void Revoke(ClientFileKey fk, PrivateKey pk) throws InvalidKeyException,
        NoSuchAlgorithmException, SignatureException, ClientProtocolException, IOException
43     {
44         Gson gson = new Gson();
45         String message = gson.toJson(fk);
46         String signature = ClientCrypto.sign(pk, message);
47         SignedRequest sr = new SignedRequest(message, signature);
48         String signedJSON = gson.toJson(sr);
49         HttpClient httpClient = HttpClientBuilder.create().build();
50         HttpPost post = new HttpPost("http://80.111.202.166:8000/revokefile");
51         StringEntity postingString = new StringEntity(signedJSON);
52         post.setEntity(postingString);
53         post.setHeader("Content-type", "application/json");
54         HttpResponse response = httpClient.execute(post);
55         if(response.getStatusLine().getStatusCode() != 200)
56         {
57             System.out.println("Response was not Positive:
58 "+response.getStatusLine().getStatusCode());
59         }
60         else System.out.println("Response was OK");
61     }
62 }
```

ClientFileKeyFunctions.java

```
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

```
1 package client;
2 import java.io.IOException;
3 import com.google.gson.Gson;
4 import org.apache.http.HttpResponse;
5 import org.apache.http.client.ClientProtocolException;
6 import org.apache.http.client.HttpClient;
7 import org.apache.http.client.methods.HttpGet;
8 import org.apache.http.client.methods.HttpPost;
9 import org.apache.http.entity.StringEntity;
10 import org.apache.http.impl.client.HttpClientBuilder;
11 import org.apache.http.util.EntityUtils;
12
13 class ClientUserFunctions
14 {
15     static int Register(serverUtil.User u) throws ClientProtocolException, IOException
16     {
17
18         Gson gson = new Gson();
19         HttpClient httpClient = HttpClientBuilder.create().build();
20         HttpPost post = new HttpPost("http://80.111.202.166:8000/register");
21         StringEntity postingString = new StringEntity(gson.toJson(u));
22         post.setEntity(postingString);
23         post.setHeader("Content-type", "application/json");
24         HttpResponse response = httpClient.execute(post);
25         if(response.getStatusLine().getStatusCode()!=200)
26         {
27             System.out.println("Response was not Positive: "+response.getStatusLine
28             ().getStatusCode());
29         }
30         else System.out.println("Response was OK");
31         return 0;
32     }
33     static serverUtil.User GetUser(String username) throws ClientProtocolException,
34     IOException
35     {
36         HttpClient httpClient = HttpClientBuilder.create().build();
37         HttpGet get = new HttpGet("http://80.111.202.166:8000"+"/users/"+username);
38         get.setHeader("Content-type", "application/json");
39         HttpResponse response = httpClient.execute(get);
40         String json = EntityUtils.toString(response.getEntity());
41         Gson g = new Gson();
42         return g.fromJson(json, serverUtil.User.class);
43     }
44 }
45
46
```

FileUsers.java

```
1 package client;
2
3 class FileUsers
4 {
5     String user;
6
7     FileUsers(String user) {
8         super();
9         this.user = user;
10    }
11 }
12
```

SignedRequest.java

```
1 package client;
2
3 class SignedRequest
4 {
5     public String message;
6     public String signature;
7     SignedRequest(String message, String signature) {
8         super();
9         this.message = message;
10        this.signature = signature;
11    }
12 }
13
```

InitDB.java

```
1 package initDB;
2
3 import org.apache.log4j.BasicConfigurator;
4
5 import com.rethinkdb.*;
6 import com.rethinkdb.net.Connection;
7
8
9 class InitDB
10 {
11     public static final RethinkDB r = RethinkDB.r;
12
13     public static void main(String[] args)
14     {
15         BasicConfigurator.configure();
16
17         String DBHost = "127.0.0.1";
18         Connection conn = r.connection().hostname(DBHost).port(28015).connect();
19
20         r.dbCreate("Cloud_Encryption").run(conn);
21         r.db("Cloud_Encryption").tableCreate("users").run(conn);
22         r.db("Cloud_Encryption").table("users").indexCreate("username").run(conn);
23
24         r.db("Cloud_Encryption").tableCreate("files").run(conn);
25         r.db("Cloud_Encryption").table("files").indexCreate("name").run(conn);
26         r.db("Cloud_Encryption").table("files").indexCreate("owner").run(conn);
27
28         r.db("Cloud_Encryption").tableCreate("filekeys").run(conn);
29         r.db("Cloud_Encryption").table("filekeys").indexCreate("name").run(conn);
30         r.db("Cloud_Encryption").table("filekeys").indexCreate("owner").run(conn);
31         r.db("Cloud_Encryption").table("filekeys").indexCreate("user").run(conn);
32
33         System.exit(0);
34     }
35 }
36
37
```

File.java

```

1 package server;
2
3 import java.util.HashMap;
4
5 import com.rethinkdb.RethinkDB;
6 import com.rethinkdb.gen.ast.Table;
7 import com.rethinkdb.net.Connection;
8 import com.rethinkdb.net.Cursor;
9
10 public class File
11 {
12     private static String DBHost = "127.0.0.1";
13     private static final RethinkDB r = RethinkDB.r;
14     private static Connection conn = r.connection().hostname(DBHost).port(28015).connect();
15     private static Table filetable = r.db("Cloud_Encryption").table("files");
16
17
18     static int insert(serverUtil.File f)
19     {
20         if(filetable.g("name").contains(f.name).run(conn))
21         {
22             if(filetable.g("owner").contains(f.owner).run(conn))
23             {
24                 System.out.println("Duplicate File Entry");
25                 serverUtil.File file = getFile(f.owner, f.name);
26                 f.id = file.id;
27                 filetable.insert(r.hashMap("name", f.name).with("owner",
28 f.owner).with("id", f.id).with("data", r.binary(f.data))).optArg("conflict",
29 "replace").run(conn);
30                 return 1;
31             }
32         }
33         filetable.insert(r.hashMap("name", f.name).with("owner", f.owner).with("data",
34 r.binary(f.data))).run(conn);
35         return 0;
36     }
37
38     @SuppressWarnings("rawtypes")
39     static serverUtil.File getFile(String Owner, String filename)
40     {
41         try {
42             Cursor dbRes = filetable.getAll(filename).optArg("index",
43 "name").filter(r.hashMap("owner", Owner)).run(conn);
44             HashMap m = (HashMap) dbRes.next();
45             serverUtil.File f = new serverUtil.File((String)m.get("id"),
46 (String)m.get("owner"), (String)m.get("name"), (byte[])m.get("data"));
47             return f;
48         } catch (java.util.NoSuchElementException e) {
49             System.out.println("Invalid file access");
50             System.out.println("File "+filename+" owned by "+Owner+" does not exist, perhaps
51 unauthorized access!");
52         }
53         return new serverUtil.File();
54     }
55 }

```

FileKey.java

```

1 package server;
2 import com.google.gson.Gson;
3 import com.rethinkdb.*;
4 import com.rethinkdb.gen.ast.Table;
5 import com.rethinkdb.net.Connection;
6 import com.rethinkdb.net.Cursor;
7
8 import java.util.HashMap;
9 import java.util.List;
10
11 public class FileKey
12 {
13     private static String DBHost = "127.0.0.1";
14     private static final RethinkDB r = RethinkDB.r;
15     private static Connection conn = r.connection().hostname(DBHost).port(28015).connect();
16     private static Table filekeytable = r.db("Cloud_Encryption").table("filekeys");
17
18
19     static int insert(serverUtil.FileKey fk)
20     {
21         if(filekeytable.g("name").contains(fk.name).run(conn))
22         {
23             if(filekeytable.g("owner").contains(fk.owner).run(conn))
24             {
25                 if(filekeytable.g("user").contains(fk.user).run(conn))
26                 {
27                     System.out.println("Duplicate FileKey Entry");
28                     serverUtil.FileKey f = getFileKey(fk.owner, fk.name, fk.user);
29                     fk.id = f.id;
30                     filekeytable.insert(r.hashMap("name", fk.name).with("id",
31 fk.id).with("user", fk.user).with("owner", fk.owner).with("key",
32 r.binary(fk.key))).optArg("conflict", "replace").run(conn);
33                     return 1;
34                 }
35             }
36         }
37         filekeytable.insert(r.hashMap("name", fk.name).with("user", fk.user).with("owner",
38 fk.owner).with("key", r.binary(fk.key))).run(conn);
39         return 0;
40     }
41
42     static int revoke(serverUtil.FileKey fk)
43     {
44         if(fk.user == fk.owner)
45         {
46             System.out.println("Cannot revoke own file access");
47             return 2;
48         }
49         if(filekeytable.g("name").contains(fk.name).run(conn))
50         {
51             if(filekeytable.g("owner").contains(fk.owner).run(conn))
52             {
53                 if(filekeytable.g("user").contains(fk.user).run(conn))
54                 {
55                     filekeytable.get(getFileKey(fk.owner, fk.name,
56 fk.user).id).delete().run(conn);
57                     return 0;
58                 }
59             }
60         }
61     }
62 }

```

FileKey.java

```

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",
70 "name").filter(r.hashMap("owner", owner).with("user", user)).run(conn);
71         HashMap m = (HashMap) dbRes.next();
72         serverUtil.FileKey fk = new serverUtil.FileKey((String)m.get("id"),
73 (String)m.get("user"), (String)m.get("owner"), (String)m.get("name"), (byte[])m.get("key"));
74         return fk;
75     } catch (java.util.NoSuchElementException e) {
76         System.out.println("Invalid file access");
77         System.out.println("User "+user+" does not have access to file "+filename+"
78 owned by "+owner);
79     }
80     return new serverUtil.FileKey();
81 }
82
83 @SuppressWarnings("rawtypes")
84 static List getFileUsers(String owner, String filename)
85 {
86
87     Cursor dbRes = filekeytable.getAll(filename).optArg("index",
88 "name").filter(r.hashMap("owner", owner)).pluck("user").run(conn);
89 // System.out.println(dbRes.toList());
90 // Gson gson = new Gson();
91 // System.out.println( gson.toJson(dbRes.toList()));
92     return dbRes.toList();
93 }
94
95 }
96
97 }
98
99 }
100
101 }
102
103 }
104
105 }
106
107 }
108
109 }
110
111 }
112
113 }
114
115 }
116
117 }
118
119 }
120
121 }
122
123 }
124
125 }
126
127 }
128
129 }
130
131 }
132
133 }
134
135 }
136
137 }
138
139 }
140
141 }
142
143 }
144
145 }
146
147 }
148
149 }
150
151 }
152
153 }
154
155 }
156
157 }
158
159 }
160
161 }
162
163 }
164
165 }
166
167 }
168
169 }
170
171 }
172
173 }
174
175 }
176
177 }
178
179 }
180
181 }
182
183 }
184
185 }
186
187 }
188
189 }
190
191 }
192
193 }
194
195 }
196
197 }
198
199 }
200
201 }
202
203 }
204
205 }
206
207 }
208
209 }
210
211 }
212
213 }
214
215 }
216
217 }
218
219 }
220
221 }
222
223 }
224
225 }
226
227 }
228
229 }
230
231 }
232
233 }
234
235 }
236
237 }
238
239 }
240
241 }
242
243 }
244
245 }
246
247 }
248
249 }
250
251 }
252
253 }
254
255 }
256
257 }
258
259 }
260
261 }
262
263 }
264
265 }
266
267 }
268
269 }
270
271 }
272
273 }
274
275 }
276
277 }
278
279 }
280
281 }
282
283 }
284
285 }
286
287 }
288
289 }
290
291 }
292
293 }
294
295 }
296
297 }
298
299 }
300
301 }
302
303 }
304
305 }
306
307 }
308
309 }
310
311 }
312
313 }
314
315 }
316
317 }
318
319 }
320
321 }
322
323 }
324
325 }
326
327 }
328
329 }
330
331 }
332
333 }
334
335 }
336
337 }
338
339 }
340
341 }
342
343 }
344
345 }
346
347 }
348
349 }
350
351 }
352
353 }
354
355 }
356
357 }
358
359 }
360
361 }
362
363 }
364
365 }
366
367 }
368
369 }
370
371 }
372
373 }
374
375 }
376
377 }
378
379 }
380
381 }
382
383 }
384
385 }
386
387 }
388
389 }
390
391 }
392
393 }
394
395 }
396
397 }
398
399 }
400
401 }
402
403 }
404
405 }
406
407 }
408
409 }
410
411 }
412
413 }
414
415 }
416
417 }
418
419 }
420
421 }
422
423 }
424
425 }
426
427 }
428
429 }
430
431 }
432
433 }
434
435 }
436
437 }
438
439 }
440
441 }
442
443 }
444
445 }
446
447 }
448
449 }
450
451 }
452
453 }
454
455 }
456
457 }
458
459 }
460
461 }
462
463 }
464
465 }
466
467 }
468
469 }
470
471 }
472
473 }
474
475 }
476
477 }
478
479 }
480
481 }
482
483 }
484
485 }
486
487 }
488
489 }
490
491 }
492
493 }
494
495 }
496
497 }
498
499 }
500
501 }
502
503 }
504
505 }
506
507 }
508
509 }
510
511 }
512
513 }
514
515 }
516
517 }
518
519 }
520
521 }
522
523 }
524
525 }
526
527 }
528
529 }
530
531 }
532
533 }
534
535 }
536
537 }
538
539 }
540
541 }
542
543 }
544
545 }
546
547 }
548
549 }
550
551 }
552
553 }
554
555 }
556
557 }
558
559 }
560
561 }
562
563 }
564
565 }
566
567 }
568
569 }
570
571 }
572
573 }
574
575 }
576
577 }
578
579 }
580
581 }
582
583 }
584
585 }
586
587 }
588
589 }
590
591 }
592
593 }
594
595 }
596
597 }
598
599 }
600
601 }
602
603 }
604
605 }
606
607 }
608
609 }
610
611 }
612
613 }
614
615 }
616
617 }
618
619 }
620
621 }
622
623 }
624
625 }
626
627 }
628
629 }
630
631 }
632
633 }
634
635 }
636
637 }
638
639 }
640
641 }
642
643 }
644
645 }
646
647 }
648
649 }
650
651 }
652
653 }
654
655 }
656
657 }
658
659 }
660
661 }
662
663 }
664
665 }
666
667 }
668
669 }
670
671 }
672
673 }
674
675 }
676
677 }
678
679 }
680
681 }
682
683 }
684
685 }
686
687 }
688
689 }
690
691 }
692
693 }
694
695 }
696
697 }
698
699 }
700
701 }
702
703 }
704
705 }
706
707 }
708
709 }
710
711 }
712
713 }
714
715 }
716
717 }
718
719 }
720
721 }
722
723 }
724
725 }
726
727 }
728
729 }
730
731 }
732
733 }
734
735 }
736
737 }
738
739 }
740
741 }
742
743 }
744
745 }
746
747 }
748
749 }
750
751 }
752
753 }
754
755 }
756
757 }
758
759 }
760
761 }
762
763 }
764
765 }
766
767 }
768
769 }
770
771 }
772
773 }
774
775 }
776
777 }
778
779 }
780
781 }
782
783 }
784
785 }
786
787 }
788
789 }
790
791 }
792
793 }
794
795 }
796
797 }
798
799 }
800
801 }
802
803 }
804
805 }
806
807 }
808
809 }
810
811 }
812
813 }
814
815 }
816
817 }
818
819 }
820
821 }
822
823 }
824
825 }
826
827 }
828
829 }
830
831 }
832
833 }
834
835 }
836
837 }
838
839 }
840
841 }
842
843 }
844
845 }
846
847 }
848
849 }
850
851 }
852
853 }
854
855 }
856
857 }
858
859 }
860
861 }
862
863 }
864
865 }
866
867 }
868
869 }
870
871 }
872
873 }
874
875 }
876
877 }
878
879 }
880
881 }
882
883 }
884
885 }
886
887 }
888
889 }
890
891 }
892
893 }
894
895 }
896
897 }
898
899 }
900
901 }
902
903 }
904
905 }
906
907 }
908
909 }
910
911 }
912
913 }
914
915 }
916
917 }
918
919 }
920
921 }
922
923 }
924
925 }
926
927 }
928
929 }
930
931 }
932
933 }
934
935 }
936
937 }
938
939 }
940
941 }
942
943 }
944
945 }
946
947 }
948
949 }
950
951 }
952
953 }
954
955 }
956
957 }
958
959 }
960
961 }
962
963 }
964
965 }
966
967 }
968
969 }
970
971 }
972
973 }
974
975 }
976
977 }
978
979 }
980
981 }
982
983 }
984
985 }
986
987 }
988
989 }
990
991 }
992
993 }
994
995 }
996
997 }
998
999 }

```

Server.java

```

1 package server;
2 import java.awt.BorderLayout;
3
4 import javax.swing.JFrame;
5 import javax.swing.JLabel;
6 import javax.swing.JOptionPane;
7 import javax.swing.JScrollPane;
8 import javax.swing.JTextArea;
9
10 import java.io.IOException;
11 import java.io.PrintStream;
12
13 import express.Express;
14 import serverUtil.TextAreaOutputStream;
15
16 public class Server
17 {
18     public static void main(String[] args) throws IOException
19     {
20         final JFrame frame = new JFrame();
21         frame.add( new JLabel("Server" ), BorderLayout.NORTH );
22         JTextArea ta = new JTextArea();
23         TextAreaOutputStream taos = new TextAreaOutputStream(ta);
24         PrintStream ps = new PrintStream(taos);
25         System.setOut(ps);
26         System.setErr(ps);
27         ta.setEditable(false);
28         frame.add(new JScrollPane(ta));
29         frame.pack();
30         frame.setVisible(true);
31         frame.setSize(800, 600);
32
33         frame.addWindowListener(new java.awt.event.WindowAdapter() {
34             @Override
35             public void windowClosing(java.awt.event.WindowEvent windowEvent) {
36                 if (JOptionPane.showConfirmDialog(frame,
37                     "Are you sure you want to close the Server?", "Close Server?",
38                     JOptionPane.YES_NO_OPTION,
39                     JOptionPane.QUESTION_MESSAGE) == JOptionPane.YES_OPTION){
40                     System.exit(0);
41                 }
42             }
43         });
44         frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
45
46         Express app = new Express();
47         try {
48             System.out.println("Starting up Server...");
49             app.bind(new ServerBindings()); // See class below
50             app.listen(8000);
51         } catch (java.lang.ExceptionInInitializerError e) {
52             System.out.println("Server Refused Connection. Shutting Down.");
53             System.exit(0);
54         }
55     }
56
57 }
58

```


ServerBindings.java

```

1 package server;
2
3 import java.io.IOException;
4 import java.io.InputStreamReader;
5 import java.io.UnsupportedEncodingException;
6 import java.security.InvalidKeyException;
7 import java.security.KeyFactory;
8 import java.security.NoSuchAlgorithmException;
9 import java.security.SignatureException;
10 import java.security.spec.InvalidKeySpecException;
11 import java.security.spec.X509EncodedKeySpec;
12 import java.util.List;
13
14 import com.google.gson.Gson;
15 import com.google.gson.JsonIOException;
16 import com.google.gson.JsonObject;
17 import com.google.gson.JsonParser;
18 import com.google.gson.JsonSyntaxException;
19 import express.DynExpress;
20 import express.http.RequestMethod;
21 import express.http.request.Request;
22 import express.http.response.Response;
23 import express.utils.Status;
24
25 class ServerBindings {
26
27     @DynExpress(context= "/register", method = RequestMethod.POST) // Default is
context="/" and method=RequestMethod.GET
28     public void register(Request req, Response res) throws IOException {
29         System.out.println("Received Register Request!");
30         JsonParser jsonParser = new JsonParser();
31         JsonObject jsonObject = (JsonObject)jsonParser.parse(
32             new InputStreamReader(req.getBody(), "UTF-8"));
33         Gson gson = new Gson();
34         String json = gson.toJson(jsonObject);
35         serverUtil.User user = gson.fromJson(json,serverUtil.User.class);
36         if(User.insert(user)!=0)res.send("Duplicate User Entry!");
37         else res.send("User Registered!");
38     }
39
40     @DynExpress(context= "/uploadfile", method = RequestMethod.POST) // Only context is
defined, method=RequestMethod.GET is used as method
41     public void uploadFile(Request req, Response res) throws JsonIOException,
JsonSyntaxException, UnsupportedEncodingException, InvalidKeyException, SignatureException,
NoSuchAlgorithmException, InvalidKeySpecException {
42         System.out.println("Received Upload Request!");
43         JsonParser jsonParser = new JsonParser();
44         JsonObject jsonObject = (JsonObject)jsonParser.parse(
45             new InputStreamReader(req.getBody(), "UTF-8"));
46         Gson gson = new Gson();
47         String message = jsonObject.get("message").getAsString();
48         String sig = jsonObject.get("signature").getAsString();
49
50         serverUtil.File file = gson.fromJson(message, serverUtil.File.class);
51         serverUtil.User u = User.GetUser(file.owner);
52         if(!serverUtil.Crypto.verify(KeyFactory.getInstance("RSA").generatePublic(new
X509EncodedKeySpec(u.PubKey)), message.getBytes(), sig))
53         {
54             System.out.println("Could not Verify Signature!");
55             res.setStatus(Status.valueOf(403));
56             res.send("Could not Verify Signature!");
57         }
58     }
59 }

```

ServerBindings.java

```

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)

```

ServerBindings.java

```

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,
126     JsonSyntaxException, UnsupportedEncodingException {
127     serverUtil.User u = User.getUser(req.getParam("username"));
128     Gson gson = new Gson();
129     String json = gson.toJson(u);
130     res.send(json);
131 }
132 @DynExpress(context = "/users/:username/:filename", method = RequestMethod.GET) // Both
133     defined
134     public void getFile(Request req, Response res) {
135         serverUtil.File f = File.getFile(req.getParam("username"), req.getParam
136             ("filename"));
137         if(f.name.equals(""))
138         {
139             res.setStatus(Status.valueOf(403));
140             res.send("Failed Getting File, does not Exist.");
141         }
142         else
143         {
144             Gson gson = new Gson();
145             String json = gson.toJson(f);
146             res.send(json);
147         }
148     }
149     @SuppressWarnings("rawtypes")
150     @DynExpress(context = "/users/:username/:filename/users", method = RequestMethod.GET)
151     // Both defined
152     public void getFileUsers(Request req, Response res) {
153         List l = FileKey.getFileUsers(req.getParam("username"), req.getParam("filename"));
154         Gson gson = new Gson();
155         String json = gson.toJson(l);
156         res.send(json);
157     }
158     @DynExpress(context = "/users/:username/:filename/key/:user", method =
159     RequestMethod.GET) // Both defined
160     public void getFileKey(Request req, Response res) {
161         serverUtil.FileKey fk = FileKey.getFileKey(req.getParam("username"), req.getParam
162             ("filename"), req.getParam("user"));
163         if (fk.name.equals(""))
164         {
165             res.setStatus(Status.valueOf(403));
166             res.send("Failed Getting FileKey, unauthorized access.");
167         }
168         else
169         {
170             Gson gson = new Gson();
171             String json = gson.toJson(fk);

```

ServerBindings.java

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

User.java

```

1 package server;
2
3 import com.google.gson.Gson;
4 import com.rethinkdb.*;
5 import com.rethinkdb.gen.ast.Table;
6 import com.rethinkdb.net.Connection;
7 import com.rethinkdb.net.Cursor;
8
9 public class User
10 {
11     private static String DBHost = "127.0.0.1";
12     private static final RethinkDB r = RethinkDB.r;
13     private static Connection conn = r.connection().hostname(DBHost).port(28015).connect();
14     private static Table userTable = r.db("Cloud_Encryption").table("users");
15
16     public static int insert(serverUtil.User u)
17     {
18         if(userTable.g("username").contains(u.username).run(conn))
19         {
20             System.out.println("Duplicate User Entry");
21             return 1;
22         }
23         else
24         {
25             serverUtil.DBUser user = new serverUtil.DBUser(u.id,u.username,u.PubKey.toString
26             ());
27             userTable.insert(r.hashMap("username", user.username).with("PubKey", r.binary
28             (u.PubKey))).run(conn);
29             return 0;
30         }
31     }
32
33     @SuppressWarnings("rawtypes")
34     static serverUtil.User GetUser(String username)
35     {
36         if(userTable.g("username").contains(username).run(conn))
37         {
38             Cursor dbRes = userTable.getAll(username).optArg("index", "username").run(conn);
39             for(Object doc : dbRes)
40             {
41                 Gson gson = new Gson();
42                 String s = gson.toJson(doc);
43                 serverUtil.User u = gson.fromJson(s, serverUtil.User.class);
44                 return u;
45             }
46         }
47         else System.out.println("User does not Exist");
48         return null;
49     }
50 }

```

Crypto.java

```
1 package serverUtil;
2 import java.security.*;
3 import java.util.Base64;
4
5
6 public class Crypto
7 {
8     public static boolean verify(PublicKey pk, byte[] message, String signature) throws
        SignatureException, InvalidKeyException, NoSuchAlgorithmException
9     {
10         Signature publicSignature = Signature.getInstance("SHA1withRSA");
11         publicSignature.initVerify(pk);
12         publicSignature.update(message);
13
14         byte[] signatureBytes = Base64.getDecoder().decode(signature);
15
16         return publicSignature.verify(signatureBytes);
17     }
18 }
19
```

DBUser.java

```
1 package serverUtil;
2
3 public class DBUser
4 {
5     public DBUser(String iD, String username, String pubKey) {
6         super();
7         ID = iD;
8         this.username = username;
9         PubKey = pubKey;
10    }
11    public String ID = "gorethink:\"id,omitempty\"";
12    public String username = "gorethink:\"username\"";
13    public String PubKey = "gorethink:\"pubkey\"";
14 }
15
```

File.java

```
1 package serverUtil;
2
3 public class File
4 {
5     public String id;
6     public String owner;
7     public String name;
8     public byte[] data;
9     public File(String iD, String owner, String name, byte[] data) {
10         super();
11         id = iD;
12         this.owner = owner;
13         this.name = name;
14         this.data = data;
15     }
16     public File()
17     {
18         id = "";
19         owner = "";
20         name = "";
21         data = "".getBytes();
22     }
23
24
25 }
26
```


FileKey.java

```
1 package serverUtil;
2
3 public class FileKey
4 {
5     public String id;
6     public String user;
7     public String owner;
8     public String name;
9     public byte[] key;
10
11     public FileKey(String iD, String user, String owner, String name, byte[] key) {
12         super();
13         id = iD;
14         this.user = user;
15         this.owner = owner;
16         this.name = name;
17         this.key = key;
18     }
19
20
21
22     public FileKey()
23     {
24         id = "";
25         user = "";
26         owner = "";
27         name = "";
28         key = "".getBytes();
29     }
30
31 }
32
```

TextAreaOutputStream.java

```
1 package serverUtil;
2
3 import java.io.IOException;
4 import java.io.OutputStream;
5 import javax.swing.JTextArea;
6
7 public class TextAreaOutputStream extends OutputStream {
8     private JTextArea textControl;
9     public TextAreaOutputStream( JTextArea control ) {
10         textControl = control;
11     }
12     public void write( int b ) throws IOException {
13         textControl.append( String.valueOf( ( char )b ) );
14     }
15 }
16
```

User.java

```
1 package serverUtil;
2
3 public class User
4 {
5     public String id;
6     public String username;
7     public byte[] PubKey;
8
9     public User(String username, byte[] pubKey) {
10         super();
11         this.username = username;
12         PubKey = pubKey;
13     }
14 }
15
```