JSONCrypt

JSON ENCRYPTION AND DECRYPTION

Sudipta Saha | Sample code | 04/07/2017

Contents

Pre installation packages required.

User needs to include org.json.jar and SmpleCipher.jar in order to use JSONCrypt.jar

Encryption Packs.

The JSONCrypt class encompasses two major encryption standards. They are :

- RSA Encryption.
- AES Encryption.

ABOUT RSA ENCRYPTION

- It is an asymmetric cryptographic algorithm.
- RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described it in 1978.
- For learning the encryption method please visit : https://simple.wikipedia.org/wiki/RSA_(algorithm)

ABOUT AES ENCRYPTION

- It is a symmetric cryptographic algorithm
- AES stands for Advanced Encryption Standard.
- AES is a subset of the Rijndael cipher[6] developed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen.
- For learning the encryption method please visit : https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

Required Packages and their description.

JSONCrypt.jar includes three major packages. Users need to be aware of the usage of stored classes before they can use the functionalities.

Package JSONCrypt –

Class Encrypt –

Method AESEncrypt-

Overloaded function that deals with AES encryption of user data. In general, the 128 bit AES key is encrypted by RSA encryption, however users may implement interfaces to define their preferred encryption stands.

Method RSAEncrypt-

Simpler and faster encryption of user data, however less secure as compared to AES encryption. Involves only RSA encryption of user data.

Class Decrypt –

Method AESDecrypt-

Overloaded function that deals with AES decryption of user data. In general, the 128 bit AES key is encrypted by RSA encryption, however users may implement interfaces to define their preferred decryption stands. Decrypts AES encrypted data.

Method RSADecrypt-

Simpler and faster decryption of user data, however less secure as compared to AES encryption standards. Involves only RSA decryption of user data from RSA encrypted data.

Interface CustomEncryption –

As mentioned already, users may implement this interface in order to use their own standards of 128 bit AES key encryption.

Interface CustomDecryption –

Equivalent decryption class for user specified AES key encryption.

Package JSONCrypt.AESCrypto –

Class AESKey –

Contains methods to create an object to 128 bit AES key from preferred binary, hexadecimal, alphanumeric or even ascii input.

Package JSONCrypt.RSACrypto –

Class RSAKey –

Contains methods to create an object to RSA key. The encryption (public key), decryption (private key) and N key are machine generated.

Client – Server communication.

In general we will make frequently make use of JSON encoded data transaction between client and server. In practical scenario, server is never aware of client as in individual.

REQUEST: Client REQUESTS Server for data.

RESPONSE: Server RESPONDS Client with data.

CYCLE: One complete cycle of client – server communication is called cycle.

TYPES OF COMMUNICATION:

Client pokes server for data and server replies and then forgets client. Hence, encryption of data may occur in following ways:

Scenario 1. Client requests Server for data where encryption of response from Server is essential but request from client can be warded off from encryption.

Scenario 2. Client requests Server for data where encryption of request from Client is essential but server response is not required to be encrypted.

Scenario 3. Client requests Server for data where encryption of both request and response is essential.

Dealing with request and response.

In case of scenario 1.

- We first ping the server with the clients RSA encryption key and N key.
- Server stores the keys.
- End of key exchange cycle.
- Second we ping the server again asking for data, server encrypts the data using any encryption standard, and then returns data to client.
- Client is able to decrypt data using RSA decryption key.
- End of communication cycle.

In case of scenario 2.

- We first ping the server and server responds with its own RSA encryption key and N key.
- Client stores the server keys.
- End of key exchange cycle.

- Second we ping the server again asking for data, where the request is encrypted using any encryption standard.
- Server is able to get the data from client by decrypting the data using its RSA decryption key.
- Server replies client with required data.
- End of communication cycle.

In case of scenario 3.

- Client pings server with its RSA encryption key and N key, let's say client_keys.
- Server stores the client_keys.
- Server responds with its own RSA encryption key and N key, let's say server_key.
- Client stores the server_keys.
- End of key exchange cycle.
- Secondly client ping the server again asking for data, where the request is encrypted using any encryption standard. Client encrypts the data using server_keys.
- Server is able to get the data from client by decrypting the data using its RSA decryption key, or server_key.
- Thirdly server encrypts the response using client_keys.
- Client decrypts the received data from server by its own RSA decryption key or client_key.
- End of communication cycle.

Dealing with client_keys and server_keys.

When client sends RSA client_keys to server, client only sends encryption key (public key) and N key. **IT NEVER SENDS ITS DECRYPTION KEY.**

While server replies client with RSA server_keys, server only sends encryption key (public key) and N key. **IT NEVER SENDS ITS DECRYPTION KEY.**

Hence the intruder is only able to encrypt the date using the encryption key and N key, but cannot decrypt the data without decryption key, which is private.

Encryption standards.

Lastly, encryption can be of only RSA encrypted data, i.e. client and server only encrypts their data using RSA encryption standards

Or, encryption can be of both RSA AND AES, where the user data is AES encrypted using 128 bit AES key, but since the AES key has to be symmetric hence client and server both has to be aware of

the key or in case of dynamic key generation, AES key is RSA encrypted before transaction or communication.

Also, interfaces like CustomEncryption and CustomDecryption allows a user to implement their own AES key encryption and decryption standards.

Important Functions.

ENCRYPTION STANDARDS.

RSA and AES Encryption

```
String ciphertext = JSONCrypt.Encrypt.AESEncrypt(aESKey, auth,
server_keys);
```

Only RSA Encryption

```
String cipherText=JSONCrypt.Encrypt.RSAEncrypt(data, rSAKeys);
```

AES Encryption with CustomEncryption

```
ClassThatImplementsCustomEncryption c=new
ClassThatImplementsCustomEncryption();

String plainText=JSONCrypt.Decrypt.AESDecrypt(cipherText, c);
```

DECRYPTION STANDARDS.

RSA and AES Decryption

```
String plainText =
JSONCrypt.Decrypt.AESDecrypt(postResponse.toString(),
client_RSAKeys);
```

Only RSA Decryption

```
String plainText = JSONCrypt.Decrypt.RSADecrypt(cipherText,
rSAKeys);
```

AES Decryption with CustomDecryption

```
ClassThatImplementsCustomEncryption c=new
ClassThatImplementsCustomEncryption();


String plainText=JSONCrypt.Decrypt.AESDecrypt(cipherText, c);
```

Sample code.

SCENARIO 1.

Client code.

- Client creates RSA keys, let's name them client_keys.
-
```
RSAKeys rSAKeys=new RSAKeys();
```

- Client creates a POST request to Server asking Server to create a connection.

```
// get RSA encryption key and N key of client_keys.
                String keys=rSAKeys.getJsonEncodedRSAKey();


     // post request URL

              String url =
"http://localhost:8084/CoverWeb/Send_Encrypt_Only";
          URL obj = new URL(url);

//Post request connection

            HttpURLConnection con = (HttpURLConnection)
obj.openConnection();
            con.setRequestMethod("GET");
            con.setRequestProperty("User-Agent", USER_AGENT);
            con.setDoInput(true);
            con.setDoOutput(true);



//Create data stream to send POST parameters (if any) in our
case we send the keys.


          String urlParameters = "keys="+keys;
```

```
                DataOutputStream wr = new


DataOutputStream(con.getOutputStream());
              wr.writeBytes(urlParameters);
           wr.flush();
              wr.close();


//get the post request response code. 200 is OK.


                  int responseCode = con.getResponseCode();
                  System.out.println("\nSending 'POST' request
to URL : " + url);
              System.out.println("Response Code : " +
responseCode);



// Receive server response using BufferedReader.

                  BufferedReader in = new BufferedReader(
                     new
InputStreamReader(con.getInputStream()));
              String inputLine;

              StringBuffer postResponse = new StringBuffer();

              while ((inputLine = in.readLine()) != null) {
                   postResponse.append(inputLine);
              }
              in.close();



//after receiving server response decrypt the data and print.

      out.print(JSONCrypt.Decrypt.AESDecrypt(postResponse.toSt
ring(), rSAKeys));
```

Server code.

- Server receives the client_keys and stores them.

```
// server get the request parameters. The parameter key has the
client_keys. Server stores it.

String keys=request.getParameter("keys");

            System.out.print("\n keys : "+keys);

           if(keys!=null){

           try {

                org.json.JSONObject jbj=new JSONObject(keys);

                String data="I AM THE DATA TO BE SENT";

      // Server generates AES encryption key.


AESKey aESKey=AESKey.randomToKey();
```

- Server uses AES encryption to encrypt the data and send it back to client.

```
out.print(JSONCrypt.Encrypt.AESEncrypt(aESKey, data,jbj));
```

- End of communication.

SCENARIO 2.

Client code:

```
//Create the client RSA keys
```

```
final RSAKeys client_RSAKeys=new RSAKeys();



//Get the tranmission ready client_keys, we send the keys to
server here

String keys=client_RSAKeys.getJsonEncodedRSAKey();



//Set up POST request


                String url =
"http://localhost:8084/CoverWeb/Server_Send_And_Receive";

                URL obj = new URL(url);

                HttpURLConnection con = (HttpURLConnection)
obj.openConnection();

                con.setRequestMethod("POST");

                con.setRequestProperty("User-Agent",
USER_AGENT);


                con.setDoInput(true);

                con.setDoOutput(true);


//Parameter op stands for operation. We inform the server that
we are exchanging keys first. Hence we send the keys along with
the request.
```

```
                String urlParameters =
"op=createCon&keys="+keys;

                DataOutputStream wr = new
DataOutputStream(con.getOutputStream());

                wr.writeBytes(urlParameters);

                wr.flush();

                wr.close();

                int responseCode = con.getResponseCode();

                System.out.println("\nSending 'POST' request to
URL : " + url);


//Server responses is stored in string buffer postResponse




                System.out.println("Response Code : " +
responseCode);

                BufferedReader in = new BufferedReader(

                        new
InputStreamReader(con.getInputStream()));

                String inputLine;


                StringBuffer postResponse = new StringBuffer();


                while ((inputLine = in.readLine()) != null) {

                    postResponse.append(inputLine);

                }

                in.close();
```

```
                System.out.print("Server Keys :
"+postResponse.toString());

server_rSAKey=postResponse.toString();

               out.print("\nConnection created.");


//We extract the server_keys from server response and then use
it to encrypt our data. Here we use both RSA and AES encryption.


//JSON data that we will send to server.

             org.json.JSONObject server_keys= new
JSONObject(server_rSAKey);

             org.json.JSONObject auth= new JSONObject();

             auth.put("username", "subho040995@gmail.com");

             auth.put("password", "XXX");



//AES key generation


            AESKey aESKey=AESKey.randomToKey();


//Cipher text is created here.




              String
ciphertext=JSONCrypt.Encrypt.AESEncrypt(aESKey, auth,
server_keys);


//We send the encrypted data to server.
```

```
            url =
"http://localhost:8084/CoverWeb/Server_Send_And_Receive";

                obj = new URL(url);

                con = (HttpURLConnection) obj.openConnection();

                con.setRequestMethod("POST");

                con.setRequestProperty("User-Agent",
USER_AGENT);

                con.setDoInput(true);

                con.setDoOutput(true);



//Here op is set to exchangeData, since we are exchanging the
encrypted data now. We attach the cipher text along the
exchangeData operation.


                urlParameters =
"op=exchangeData&data="+ciphertext;

                wr = new
DataOutputStream(con.getOutputStream());

                wr.writeBytes(urlParameters);

                wr.flush();

                wr.close();

                responseCode = con.getResponseCode();

                System.out.println("\nSending 'POST' request to
URL : " + url);

                System.out.println("Response Code : " +
responseCode);

                in = new BufferedReader(

                        new
InputStreamReader(con.getInputStream()));
```

```
                inputLine="";



//Server response is stored into postResponse
                postResponse = new StringBuffer();


                while ((inputLine = in.readLine()) != null) {
                    postResponse.append(inputLine);
                }
                in.close();
                out.print(postResponse.toString());


                if(!postResponse.toString().trim().equals("")){



//We receive the server encrypted data here, and decrypt the
cipher text here.
                String
plainText=JSONCrypt.Decrypt.AESDecrypt(postResponse.toString(),
client_RSAKeys);


//Evaluate the server response.
                 out.print("Server says : Successful login :
"+plainText);
                }
                else{
                   out.print(" Null Response :p");
                }
```

Server code:

```
// Server checks if the operation is to createConnection
if(request.getParameter("op").equals("createCon")){
// Store the client encryption key.
            String keys=request.getParameter("keys");
            System.out.print("\n Client keys : "+keys);
            if(keys!=null){



        client_rAKeys=keys;
//Create new server_keys.


              server_rSAKeys=new RSAKeys();

//Respond to the client with the server_keys.
             out.print(server_rSAKeys.getJsonEncodedRSAKey());


            }
            }

//If operation is to exchangeData
          else
if(request.getParameter("op").equals("exchangeData")){
```

```
//we receive the encrypted data.
String ciphertext=request.getParameter("data");


             if(server_rSAKeys!=null){
//Decrypt the data here.


             String
plaintext=JSONCrypt.Decrypt.AESDecrypt(ciphertext,
server_rSAKeys);
             org.json.JSONObject auth=new
JSONObject(plaintext);
             org.json.JSONObject reply=new JSONObject();



//Check for valid authentication from decrypted data.


if(auth.getString("username").equals("subho040995@gmail.com") &&
auth.getString("password").equals("XXX")){


//Encrypt server response.


                org.json.JSONObject client_key= new
JSONObject(client_rAKeys);
                 System.out.print("\n Client rsa key at server
: "+client_rAKeys);
                  reply.put("success", "true");
```

```
//Send client the encrypted data.



out.print(JSONCrypt.Encrypt.AESEncrypt(AESKey.randomToKey(),repl
y, client_key));
                }
                else{
                    reply.put("success", "false");
                }
                  //  AESKey aESKey=AESKey.randomToKey();




                }
                else{
                  System.out.print("\nServer says : null rsakey");


                }
```