About

The JSONCrypt library is specifically used for encrypting and decrypting data that is sent and received between a client and a server. The library makes use of 128 bit AES encryption of data, followed by either RSA encryption or developer specified custom encryption of the AES key. This grants the user absolute right to control and manage the security of their data. Further, a user is entitled to various methods of creating an AES key for encryption.

Library users include both **J2EE** and **Android** developers working extensively on server request and response.

Soon to expect: Support for **PHP** and **Javascript**.

Make sure you include **org.json.jar** and **SimpleCipher-binaries.jar**.

Communication scenarios:

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.

Developer's approach:

The communication cycle can be partitioned into two parts as follows:

Create connection: Initially we need to create the RSA keys and then exchange the same between client and server. Data security is an absolute essentiality, hence developers need not to worry about leak of RSA keys since RSA keys are asymmetric in nature. We use encryption key (public) to encrypt data and hence is transmitted over the channel. The decryption key (private) remains with the host.

Exchange data: Once the keys have been exchanged, we can make second request to server and exchange the encrypted data. Since receiver will have the host key, so the data can be then decrypted at receiver end.

Sample Code:

Following codes are both for **Android** and **J2EE** developers.

Create connection:

Creating RSA key:

```
RSAKeys rSAKeys=new RSAKeys();
String keys=rSAKeys.getJsonEncodedRSAKey();
```

From Client: send client RSA Keys:

```
String url = "<... Server address ...>";
URL obj = new URL(url);
HttpURLConnection con = (HttpURLConnection) obj.openConnection();
con.setRequestMethod("GET");
con.setRequestProperty("User-Agent", USER_AGENT);
con.setDoInput(true);
con.setDoOutput(true);
String urlParameters = "operation=createCon"+"&keys="+keys;
DataOutputStream wr = new DataOutputStream(con.getOutputStream());
wr.writeBytes(urlParameters);
wr.flush();
wr.close();
```

From Client: receive server RSA Keys:

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

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

From Server: Receive client RSA Keys:

```
if(request.getParameter("op").equals("createCon")){
       String keys=request.getParameter("keys");
       if(keys!=null){
              client_rAKeys=keys;
```

From Server: Send server RSA Keys:

```
        server_rSAKeys=new RSAKeys();
        out.print(server_rSAKeys.getJsonEncodedRSAKey());
```

Exchange Data:

Creating AES key:

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

From Client: send encrypted data:

```
String ciphertext=JSONCrypt.Encrypt.AESEncrypt(aESKey, auth,
server_keys);
url = "<... Server address ...>";
```

```
obj = new URL(url);
con = (HttpURLConnection) obj.openConnection();
con.setRequestMethod("POST");
con.setRequestProperty("User-Agent", USER_AGENT);
con.setDoInput(true);
con.setDoOutput(true);

urlParameters = "operation=exchangeData"+"&data="+ciphertext;
wr = new DataOutputStream(con.getOutputStream());
wr.writeBytes(urlParameters);
wr.flush();
wr.close();
```

From Client: receive encrypted data:

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

StringBuffer postResponse = new StringBuffer();
while ((inputLine = in.readLine()) != null) {
    postResponse.append(inputLine);
}
in.close();
if(!postResponse.toString().trim().equals("")){

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

From Server: receive encrypted data:

```
if(request.getParameter("op").equals("exchangeData")){
```

```
String ciphertext=request.getParameter("data");

if(server_rSAKeys!=null){

String plaintext=JSONCrypt.Decrypt.AESDecrypt(ciphertext,
server_rSAKeys);
```

From Server: send encrypted data:

```
String
cipherText=JSONCrypt.Encrypt.AESEncrypt(AESKey.randomToKey(),reply,
client_key);

out.print(cipherText);
```

Custom AES Key Encryption:

You may implement built in interfaces CustomEncryption and CustomDecryption and override methods customEncryption(String) and customDecryption(String) in order to customize the AES key encryption and decryption. Assuming you have implemented the interfaces and overridden the methods, the AES encryption can then be carried out as follows:

Create object of the class that implements the interfaces respectively,

Encryption

```
class MyAESEncryption implements JSONCrypt.CustomEncryption{

.

.

.

MyAESEncryption myAESEncryption = new MyAESEncryption(...);

JSONCrypt.AESCrypto.AESKey aESKey=AESKey.randomToKey();

String cipherText=JSONCrypt.Encrypt.AESEncrypt(aESKey, data,
myAESEncryption);

.

.

.
```

Decryption

```
class MyAESDecryption implements JSONCrypt.CustomDecryption{

.

.

.

MyAESDecryption myAESEncryption = new MyAESDecryption(...);

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

.

.

.
```