```
1  ------------------SymmetricEncryption.java---------------
2  package com.ameya.test;
3
4  import java.security.InvalidKeyException;
5  import java.security.NoSuchAlgorithmException;
6  import java.security.SecureRandom;
7  import java.util.Base64;
8
9  import javax.crypto.BadPaddingException;
10 import javax.crypto.Cipher;
11 import javax.crypto.IllegalBlockSizeException;
12 import javax.crypto.KeyGenerator;
13 import javax.crypto.NoSuchPaddingException;
14 import javax.crypto.SecretKey;
15
16 public class SymmetricEncryption {
17
18     public static void main(String[] args) throws NoSuchAlgorithmException,
       NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException,
       BadPaddingException {
19         KeyGenerator keyGenObj=KeyGenerator.getInstance("AES");
20         keyGenObj.init(new SecureRandom());
21         SecretKey mySecretKey=keyGenObj.generateKey();
22         System.out.println(mySecretKey.getAlgorithm());
23         System.out.println(mySecretKey.getFormat());
24         Cipher cipher=Cipher.getInstance("AES/CBC/PKCS5Padding");
25         cipher.init(Cipher.ENCRYPT_MODE,mySecretKey);
26         byte data[]=cipher.doFinal("Ameya Joshi".getBytes());
27         System.out.println(data);
28         System.out.println(new String(Base64.getEncoder().encode(data)));
29     }
30
31 }
32 ------------------------ASymmetricEncryption.java-----------------
33 package com.ameya.test;
34
35 import java.security.KeyPair;
36 import java.security.KeyPairGenerator;
37 import java.security.PrivateKey;
38 import java.security.PublicKey;
39 import java.util.Base64;
40
41 import javax.crypto.Cipher;
42
43 public class AsymmetricEncryption {
44
45     public static void main(String[] args) throws Exception{
46         //Can we generate a key pair programaticaly = YES
```

```
47          //Can we read already generated keypair programatically = YES
48          //Can we read a public key programatically = YES
49          //Can we read private key programatically = YES (Provided You have the
            password)
50          KeyPairGenerator keyPairGen=KeyPairGenerator.getInstance("RSA");
51          keyPairGen.initialize(2048);
52          KeyPair keyPair=keyPairGen.generateKeyPair();
53          PrivateKey privateKey=keyPair.getPrivate();
54          PublicKey publicKey=keyPair.getPublic();
55          Cipher cipher=Cipher.getInstance("RSA/ECB/PKCS1Padding");
56          //Encrypt using public key
57          cipher.init(Cipher.ENCRYPT_MODE, publicKey);
58          byte[] data = cipher.doFinal("HElloWorldCrypto".getBytes());
59          System.out.println(data);
60          System.out.println(new String(Base64.getEncoder().encode(data)));
61          //private key
62          cipher.init(Cipher.DECRYPT_MODE, privateKey);
63          byte data1[]=cipher.doFinal(data);
64          System.out.println("-------------------------------------");
65          System.out.println(data1);
66          System.out.println(new String(Base64.getEncoder().encode(data1)));
67      }
68
69  }
70  ------------------------CryptoHelper.java----------------
71  package com.ameya.test;
72
73  import java.io.FileInputStream;
74  import java.security.KeyStore;
75  import java.security.PrivateKey;
76  import java.security.PublicKey;
77  import java.security.Signature;
78  import java.util.Base64;
79
80  import javax.crypto.Cipher;
81  //keytool -genkeypair -alias mykeystore -keyalg RSA -keysize 2048 -keypass
    123456 -validity 100 -storetype JKS -keystore
    c:\work\sapient\keystore\mkeystore.jks -storepass 123456
82  public class CryptoHelper {
83
84      public static String sign(String data) throws Exception{
85          Signature signature=Signature.getInstance("SHA256withRSA");
86          signature.initSign(getPrivateKey());
87          signature.update(data.getBytes());
88          return new String(Base64.getEncoder().encode(signature.sign()));
89      }
90      public static boolean validateSignature(String data,String signedData)throws
        Exception{
```

```
91          Signature signature=Signature.getInstance("SHA256withRSA");
92          signature.initVerify(getPublicKey());
93          signature.update(data.getBytes());
94          return signature.verify(Base64.getDecoder().decode(signedData.getBytes()));
95      }
96      public static String encrypt(String plainText) throws Exception{
97          Cipher cipher=Cipher.getInstance("RSA/ECB/PKCS1Padding");
98          cipher.init(Cipher.ENCRYPT_MODE, getPublicKey());
99          cipher.update(plainText.getBytes());
100         byte[] secretData=cipher.doFinal();
101         return new String(Base64.getEncoder().encode(secretData));
102     }
103     public static String decrypt(String encodedSecret)throws Exception{
104         Cipher cipher=Cipher.getInstance("RSA/ECB/PKCS1Padding");
105         cipher.init(Cipher.DECRYPT_MODE, getPrivateKey());
106         byte[]
            decryptedData=cipher.doFinal(Base64.getDecoder().decode(encodedSecret));
107         return new String(decryptedData);
108     }
109     private static PrivateKey getPrivateKey() {
110         KeyStore keyStore=null;
111         PrivateKey key=null;
112         try {
113             keyStore=KeyStore.getInstance("JKS");
114             keyStore.load(new
              FileInputStream("c:/work/sapient/keystore/mkeystore.jks"),
115                     "123456".toCharArray());
116             KeyStore.ProtectionParameter prameter=new
              KeyStore.PasswordProtection("123456".toCharArray());
117             key=(PrivateKey)keyStore.getKey("mykeystore", "123456".toCharArray());
118         }catch(Exception e) {
119             e.printStackTrace();//Handle Your Exceptions here
120         }
121         return key;
122     }
123     private static PublicKey getPublicKey() {
124         PublicKey key=null;
125         KeyStore keyStore=null;
126         try {
127             keyStore=KeyStore.getInstance("JKS");
128             keyStore.load(new
              FileInputStream("c:/work/sapient/keystore/mkeystore.jks"),
129                     "123456".toCharArray());
130             key=keyStore.getCertificate("mykeystore").getPublicKey();
131         }catch(Exception e) {
132             //handle the exceptions here
133         }
134         return key;
```

```
135     }
136     public static void main(String[] args)throws Exception {
137         String signedData=CryptoHelper.sign("Ameya Joshi Signature");
138         System.out.println(signedData);
139         boolean isValid=CryptoHelper.validateSignature("Ameya Joshi Signature",
            signedData);
140         System.out.println(isValid);
141     }
142 }
143
```