Com S / Cpr E 559 Spring 2020

Assignment 3: Programming Practice on Basic Crypto

Due: **Saturday, February 29**

In this assignment, you are **required** to develop three Java programs, an AsymmetricKeyProducer, a Server and a Client.

The **AsymmetricKeyProducer** program can randomly generate a pair of RSA public and private keys. It should work as follows:

1. It runs with three arguments provided by its user:
    a. The file to store the public key; and
    b. The file to store the private key.
2. It randomly generates a pair of public and private keys each with 2048 bits.
3. It stores the keys to the user-specified files, respectively. (Refer to and feel free to reuse the example code in the appendix.)

The **Client program** should work as follows:

1. It runs with the following arguments:
    a. The Server's IP address;
    b. The Server's port number;
    c. A file containing the client's private key; and
    d. A file containing the server's public key.
2. It creates a socket and connects to the Server.
3. It generates a random number (in 128 bits) as symmetric key K for AES algorithm and sends the key to the server. The key K (in byte array) should be encrypted with the public key of the server, and sent in ciphertext, together with the client's digital signature of the key. The client should print out the ciphertext and the signature.
4. It receives an encrypted file (note: the file's plaintext is encrypted with AES algorithm using key K and CBC mode; the encrypted file includes an IV) from the server.
5. The client then decrypts the file with K and output the plaintext.

The **Server program** should work as follows:

1. It runs with the following arguments:
    a. The Server's port number;
    b. A file containing the server's private key;
    c. A file containing the client's public key; and
    d. A text file.
2. It creates a socket and waits for the client to connect. Once the client connects, it works as follows.
3. It receives from the client a symmetric key K that is encrypted with its own public key, along with a signature produced by the client.
4. It decrypts the ciphertext to get K in plaintext, and verify the signature. If the verification succeeds, it outputs K in plaintext; otherwise, it outputs an error message and stop.
5. It encrypts the text file provided by its user with AES algorithm using key K and CBC mode, and outputs the ciphertext.
6. It sends the ciphertext (note: including IV) to the client.

Submission:

1. Your submission should contain AsymmetricKeyProducer.java, Client.java, and Server.java. Please keep the file names **exactly the same** as above-specified.

2. The source code should contain in-line comments for readability.
3. Make sure your submitted code compiles and runs correctly on pyrite (pyrite.cs.iastate.edu). Pyrite can be accessed using SSH tools.
4. You can have multiple submissions, but only the last one will be graded.

Appendix: Most of the skills needed for this assignment have been covered in class. To save public/private keys to files or load keys from files, refer to the following code snippets. If you are not familiar with socket programming, refer to the tutorial posted on course website.

```
//Store Public Key to file
X509EncodedKeySpec x509EncodedKeySpec =
    new X509EncodedKeySpec(publicKey.getEncoded());
FileOutputStream fos = new FileOutputStream(path);
fos.write(x509EncodedKeySpec.getEncoded());
fos.close();

//Store Private Key to file
PKCS8EncodedKeySpec pkcs8EncodedKeySpec =
    new PKCS8EncodedKeySpec(privateKey.getEncoded());
FileOutputStream fos = new FileOutputStream(path);
fos.write(pkcs8EncodedKeySpec.getEncoded());
fos.close();

public KeyPair LoadKeyPair(String path, String algorithm)
                    throws IOException, NoSuchAlgorithmException,
                    InvalidKeySpecException {

//Read Public Key from file
File filePublicKey = new File(path);
FileInputStream fis = new FileInputStream(path);
byte[] encodedPublicKey =
    new byte[(int) filePublicKey.length()];
fis.read(encodedPublicKey);
fis.close();

//Read Private Key from file
File filePrivateKey = new File(path);
fis = new FileInputStream(path);
byte[] encodedPrivateKey =
    new byte[(int) filePrivateKey.length()];
fis.read(encodedPrivateKey);
fis.close();

//Generate KeyPair
KeyFactory keyFactory = KeyFactory.getInstance(algorithm);
X509EncodedKeySpec publicKeySpec =
    new X509EncodedKeySpec(encodedPublicKey);
PublicKey publicKey =
    keyFactory.generatePublic(publicKeySpec);
PKCS8EncodedKeySpec privateKeySpec =
    new PKCS8EncodedKeySpec(encodedPrivateKey);
PrivateKey privateKey =
    keyFactory.generatePrivate(privateKeySpec);
```