Homework 2 - Solutions

In this lab, you will implement preliminary parts of the ScroogeCoin cryptocurrency. In the next lab, you will act as Scrooge and verify transactions sent to you by users and add them to your blockchain.

1. Download and install the Java Unlimited Strength Jurisdiction files from http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html. These files remove any restrictions on cryptographic strengths. Read the README file to understand why this step is needed and how to carry out the installation. If you don't install these files, you will get an "Illegal key size" exception when trying to generate keys.

2. In this lab and future labs, you must use cryptographically secure random number generation where necessary. Read the following link on generating cryptographically secure random values in Java:

https://www.cigital.com/blog/proper-use-of-javas-securerandom/

In your submission, answer the following questions:

A. Can the Java SHA1PRNG be used securely for cryptographic operations such as generate private/public key pairs?

Answer: Yes

Grading: Other answers are possible.

B. What pitfalls do programmers have be aware of when using pseudo-random number generators for cryptographic operations?

Answer: PRNGs need to be reseeded with true random seeds after enough PRNs are generated because an attacker may be able to infer the original true random seed.

Grading: Other answers are possible.

C. Why should a programmer be concerned about using `SecureRandom.getInstanceStrong()` in certain types of applications?

Answer: The method may block which isn't acceptable for interactive applications such as web applications.

Grading: Other answers are possible.

3. In ScroogeCoin, the central authority Scrooge receives transactions from users. Scrooge signs all hash pointers in the ScroogeCoin blockchain. To generate signatures, you will need to generate a private/public key pair on your computer that you can use to digitally sign transactions.

Bouncycastle (https://www.bouncycastle.org/) is a popular Java crypto library used in real world crypto systems. The lab includes a lib directory that contains the jars for this library.

Read and thoroughly understand the CryptoReference2.java file which uses crypto primitives like what Bitcoin uses. Try running the CryptoReference2 program on your computer and confirm that it completes successful without throwing exceptions. This program generates ECDSA keys. Read more about this type of cryptographic algorithm at https://en.bitcoin.it/wiki/Elliptic_Curve_Digital_Signature_Algorithm.

4. Fill in the GenerateScroogeKeyPair.java main method with code that does the following:

A. Generates a ECDSA key pair for Scrooge.

B. Stores the private key in an encrypted format on disk.

C. Store the public key in a separate, unencrypted file.

Run the class to generate the key pair for Scrooge. Name the key files as scrooge_sk.pem and scrooge_pk.pem so that it is clear who they belong to. You will use this key pair for the remaining parts of this lab as well as the next lab.

In your submission, include your code and the contents of the file containing Scrooge's public key. Do not submit your secret key. Remember never to give out your secret key and to always encrypt the secret key file when storing it on disk.

Answer:

-----BEGIN PUBLIC KEY-----

MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEQdYoKApLZuJFQ6ZDt2qavu+hPhCTpFRh

aNR3Y/Sxutj69JhfycN0pdrG129gGMSTnMCV5ssKWa75c1R4/n916A==

-----END PUBLIC KEY-----

```java
public class GenerateScroogeKeyPair {
   public static final String KEY_ALGORITHM            = "ECDSA";
   public static final String PROVIDER                 = "BC";
   public static final String CURVE_NAME               = "secp256k1";
   public static final String password              = "123456";

   public static void main(String[] args) throws Exception {
      Security.addProvider(new BouncyCastleProvider());

      SecureRandom random = SecureRandom.getInstanceStrong();
      ECGenParameterSpec ecGenSpec = new ECGenParameterSpec(CURVE_NAME);
      KeyPairGenerator keyGen_ = KeyPairGenerator.getInstance(KEY_ALGORITHM, PROVIDER);

      keyGen_.initialize(ecGenSpec, random);
      KeyPair kp = keyGen_.generateKeyPair();

      try (JcaPEMWriter wr = new JcaPEMWriter(new FileWriter("scrooge_pk.pem"))) {
         wr.writeObject(kp.getPublic());
      }

      PEMEncryptor penc = (new JcePEMEncryptorBuilder("AES-256-CFB"))
            .build(password.toCharArray());
      try (JcaPEMWriter privWriter = new JcaPEMWriter(new FileWriter("scrooge_sk.pem"))) {
         privWriter.writeObject(kp.getPrivate(), penc);
      }
   }
}
```

5. Fill in the GenerateDigitalSignature main method with code that does the following:

A. Reads Scrooge's key pair from disk

B. Generate Scrooge's digital signature for the message "Pay 3 bitcoins to Alice". Do not include the quotations in the message. Capitalization matters.

In your submission, include your code and the digital signature in hexadecimal.

Answer:
0x304502203A1D34A85E3B013A3E8E5B76E3A8EA374591CDC10B6D2D35E756FBF625C207E1022100915DF16E9
E1051940737AE62721DAEFD8A7596DA82C249F188678343D7A813DA

```
public class GenerateDigitalSignature {
   public static final String SIGNATURE_ALGORITHM     = "SHA256withECDSA";

   public static void main(String[] args) throws Throwable {
      String password = args[0];

      Security.addProvider(new BouncyCastleProvider());
      String message = "Pay 3 bitcoins to Alice";
      byte[] messageBytes = message.getBytes("UTF-8");

      PrivateKey scroogeSK = loadSecretKeyFromEncrypted(new FileReader("scrooge_sk.pem"),
password);

      Signature signature = Signature.getInstance(SIGNATURE_ALGORITHM);
      signature.initSign(scroogeSK, SecureRandom.getInstanceStrong());
      signature.update(messageBytes);
      System.out.println(DatatypeConverter.printHexBinary(signature.sign()));
   }

   public static PrivateKey loadSecretKeyFromEncrypted(Reader reader, String password) throws
IOException, NoSuchAlgorithmException, InvalidKeySpecException, PKCSException,
OperatorCreationException {
      Object o = null;
      try (PEMParser pemParser = new PEMParser(reader)) {
         o = pemParser.readObject();
      }
      PEMDecryptorProvider decProv = new
JcePEMDecryptorProviderBuilder().build(password.toCharArray());
      JcaPEMKeyConverter converter = new JcaPEMKeyConverter();
      KeyPair kp = converter.getKeyPair(((PEMEncryptedKeyPair) o).decryptKeyPair(decProv));
      return kp.getPrivate();
   }
}
```