**Homework 2**

**2. Answer the following questions:**

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

Yes. SHA1PRNG can provide sufficiently random values for cryptographic operations.

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

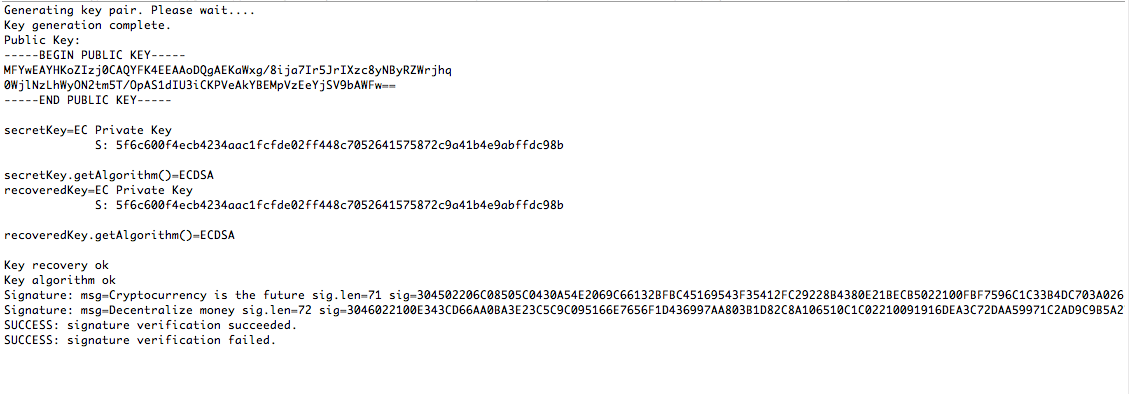
* Always specify the exact PRNG and provider that you wish to use. If you just use the default PRNG, you may end up with different PRNGs on different installations of your application that may need to be called differently in order to work properly.
* When using the SHA1PRNG, always call java.security.SecureRandom.nextBytes(byte[]) immediately after creating a new instance of the PRNG. This will force the PRNG to seed itself securely. If for testing purposes, you need predictable output, ignoring this rule and seeding the PRNG with hard-coded/predictable values may be appropriate.
* Use at least JRE 1.4.1 on Windows and at least JRE 1.4.2 on Solaris and Linux. Earlier versions do not seed the SHA1PRNG securely.
* Periodically reseed your PRNG as observing a large amount of PRNG output generated using one seed may allow the attacker to determine the seed and thus predict all future outputs.

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

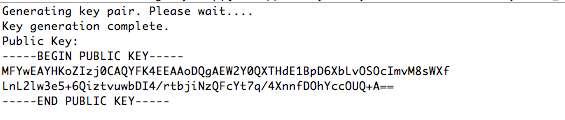
SecureRandom.getInstanceStrong() method uses the securerandom.strongAlgorithms property in the java.security file to select a SecureRandom implementation. It returns a SecureRandom object that was selected by using the algorithms/providers specified in the securerandom.strong Algorithms Security property.

However, this method additionally throws a NoSuchAlgorithmException "if no algorithm is available", which should guarantee that you never get a weak implementation back. Because of this behavior, programmers should avoid using this method in any server-side code running on Solaris/Linux/MacOS where availability is important.

3. No throwing exceptions:



4. Running result: Scrooge's public key



5. Running result: the digital signature in hexadecimal

