Homework2

2. 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?

Yes, the Java SHA1PRNG can be used securely for cryptographic operations as long as it is used appropriately.

SHA1PRNG uses a hash function and a counter, together with a seed. It generates Pseudorandom number, but it seems fit the features of random numbers (randomness, unpredictability and non-reproducibility). It is generally thought of to be secure. As it only seeds from one of the system generators during startup and therefore requires fewer calls to the kernel it is likely to be less resource intensive.

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

If a programmer call setSeed operation first instead of internal secure seeding mechanism, then the internal secure seeding mechanism is bypassed, and only the provided seed is used to generate random numbers. By bypassing the internal secure seeding mechanism of the SHA1PRNG, you may compromise the security of your PRNG output. At this time, the output of pseudorandom number generator might become predictable by attacker, then using java.security.SecureRandom may not provide the level of security that you need.

Regardless of how well the PRNG is seeded, it should not be used indefinitely without reseeding. 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.

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.

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.

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.

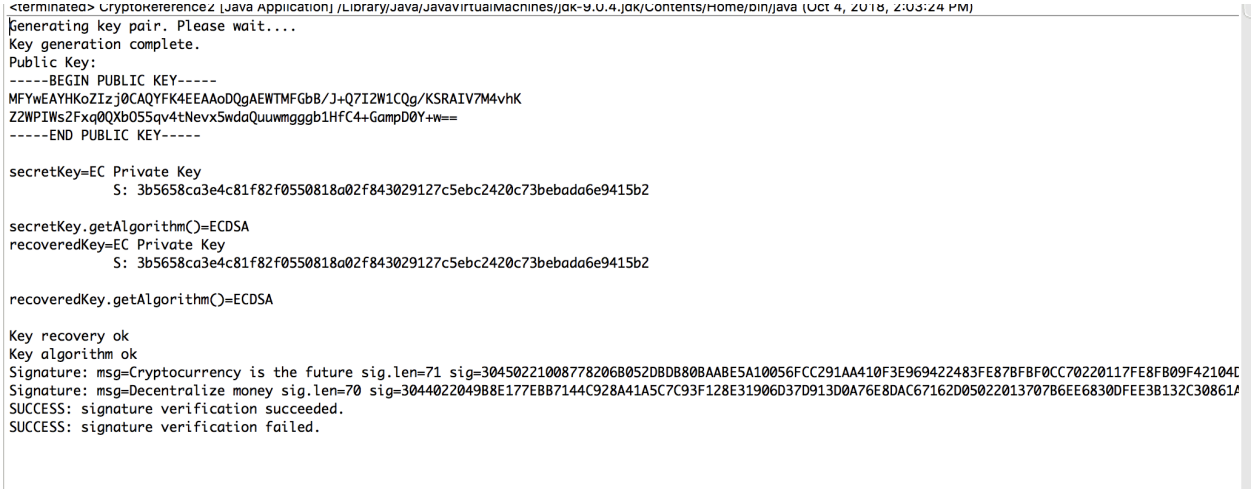
C. Why should a programmer be concerned about using SecureRandom.getInstanceStrong() in certain types of applications?

SecureRandom.getInstanceStrong() method returns a SecureRandom object that was selected by using the algorithms/providers specified in the securerandom.strongAlgorithms Security

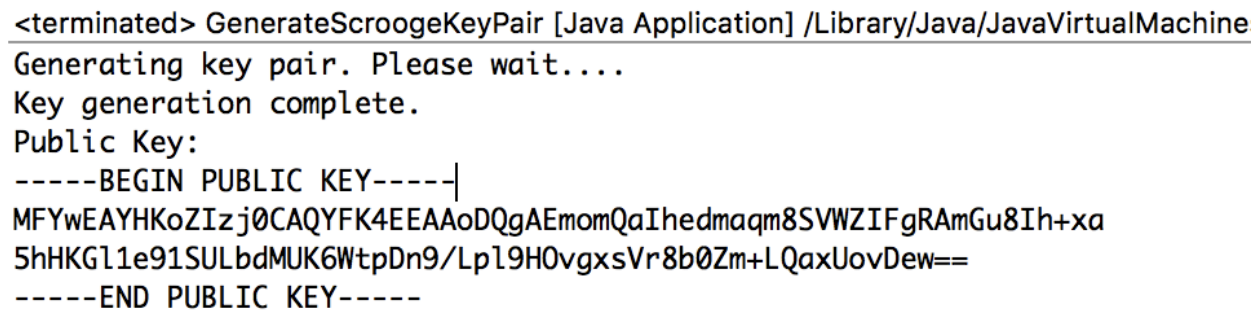
property. It will throw NoSuchAlgorithmException if no algorithm is available. This would guarantee that you will never get a weak implementation, but programmers should avoid using SecureRandom.getInstanceStrong() in any server-side code running on Solaris/Linux/MacOS where availability is important.

Due to its strong SecureRandom implementation, a programmer should also be concerned about using SecureRandom.getInstanceStrong() in some situations which require strong random values, such as when creating high-value/long-lived secrets like RSA public/private keys.

3.



4.



5.