**CS/COE 1501 Assignment 5**

**Goal:**

To get hands on experience with the use of digital signatures.

Note that the result of this project should NEVER be used for any security applications. It is purely instructive. Always use trusted and tested crypto libraries.

**High-level description:**

You will be writing two programs. The first will generate a 1024 bit RSA keypair and store the public and private keys in files named pubkey.rsa and privkey.rsa respectively. The second will generate and verify digital signatures using a SHA-256 hash. You will need to use Java's BigInteger (<http://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html>) and MessageDigest (<https://docs.oracle.com/javase/7/docs/api/java/security/MessageDigest.html>) classes to complete this project.

**Specifications:**

1. Write a program named MyKeyGen to generate a new keypair
   1. To generate a keypair, you will need to follow these steps as described in lecture:
      1. Pick P and Q to be random primes of an appropriate size to generate a 1024 bit key
      2. Generate N as P \* Q
      3. Generate PHI(N) as (P-1) \* (Q-1)
      4. Pick E such that 1 < E < PHI(N) and GCD(E, PHI(N))=1 (E must not divide PHI(N) evenly)
      5. Pick D such that D = E-1 mod PHI(N)
   2. After generating E, D, and N, save E and N to pubkey.rsa and D and N to privkey.rsa
2. Write a second program named MySign to sign files and verify signatures. This program should take in two command line arguments, a flag to specify whether it should be signing or verifying ("s" or "v"), and the file that should be signed or verified.
   1. If called to sign (e.g., "java MySign s myfile.txt") your program should:
      1. Generate a SHA-256 hash of the contents of the provided file (e.g., "myfile.txt")
      2. "decrypt" this hash value using the private key stored in privkey.rsa (i.e., raise it to the Dth power mod N)
         * Your program should exit and display an error if privkey.rsa is not found in the current directory
      3. Write out a signed version of the file (e.g., "myfile.txt.signed") that contains:
         * The contents of the original file
         * The "decrypted" hash of the original file
   2. If called to verify (e.g., "java MySign v myfile.txt.signed") your program should:
      1. Read the contents of the original file
      2. Generate a SHA-256 hash of the contents of the orignal file
      3. Read the "decrypted" hash of the original file
      4. "encrypt" this value with the contents of pubkey.rsa (i.e., raise it to the Eth power mod N)
         * Your program should exit and display an error if pubkey.rsa is not found in the current directory
      5. Compare these two hash values (the on newly generated and the one that you just "encrypted") and print out to the console whether or not the signature is valid (i.e., whether or not the values are the same).

**Submission Guidelines:**

* **DO NOT SUBMIT** any IDE package files.
* You must name your key generation program MyKeyGen.java.
* You must name your key signing/verification program MySign.java.
* You must be able to compile your game by running "javac MyKeyGen.java" and "javac "MySign.java".
* You must be able to run your key generation program with "java MyKeyGen".
* You must be able to run your signing/verification program with "java MySign s " or "java MySign v ".
* You must fill out info\_sheet.txt.
  + Be sure to remember to push the latest copy of your code back to your GitHub repository before the the assignment is due. At the deadline, the repositories will automatically be copied for grading. Whatever is present in your GitHub repository at that time will be considered your submission for this assignment.

**Additional Notes/Hints:**

* Be very careful to read over the API constructors and methods for BigInteger. Most of the functionality required for RSA key generation and encryption/decryption is already implemented in BigInteger.
* Note that creating a BigInteger based off of a byte array will truncate leading 0s in that byte array. Be sure to account for this when transforming values back and forth between BigInteger and byte[], especially when comparing hash values.
* An example of using MessageDigest to generate the SHA-256 hash of a file is provided in HashEx.java
* You may find the creation of pubkey.rsa, privkey.rsa, and signed files to be most easily accomplished through the use of ObjectOutputStreams. The format of your key files and signed files is entirely up to you, however.
* NEVER USE THIS PROJECT IN PRODUCTION CODE. This is purely instructive. Always use trusted and tested crypto libraries.

**Grading Rubric**

* Key generation
  + P and Q are generated appropriately: 10
  + N and PHI(N) computed appropriately: 5
  + E is selected appropriately: 5
  + D is selected appropriately: 15
  + Key files are generated appropriately: 10
* Signing
  + Hash is generated correctly: 5
  + Hash is "decrypted" correctly: 10
  + Signed file is generated appropriately: 10
* Verification
  + Hash is generated correctly: 5
  + Hash is "encrypted" correctly: 10
  + Signed file is generated appropriately: 10
* Assignment info sheet/submission: 5