CSE 5673 Cryptology
Final Project

Bridget Damweber
Wilson Burgos

The project was created by Bridget Damweber and Wilson Burgos. The project uses the Maven framework to build & manage dependencies. It also uses JDK 1.8 constructs and uses the JUnit test framework, gnu getopt library and Dr. Silaghi's ASN1 library.

1. Lifecycle Management

In order to manage the project you can do so using the following Maven Lifecycle commands:

- compile: compile the source code of the project
- test: test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- install: install the package into the local repository, for use as a dependency in other projects locally
- clean: cleans up artifacts created by prior builds

Once the program is compiled you can access the help to print supported options using the command:

java cs.fit.edu.rsa.RSA –h

The command output:

usage: RSA [options]

Options:

create RSA keys

-K -p public_key_file -s secret_key_file -b bits [1024 default] -y Miller_Rabin_Certainty [99.9999]

Encrypt

-e -m plaintext_file -p public_key_file -c ciphertext_file

Decrypt

-d -c ciphertext_file -s secret_key_file -m plaintext_file

2. Create RSA Keys

In order to generate a RSA private/public key pair the user issues the following command:

java cs.fit.edu.rsa.RSA -K -p public_key_file -s secret_key_file -b bits -y Miller_Rabin_certainty

The content of the files is printed to the console with one number per line, in hex. Each symbol is labelled appropriately.

3. Encryption/Decryption

In order to encrypt/decrypt a message contained in a file use the following commands:

java cs.fit.edu.rsa.RSA -e -m plaintext_file -p public_key_file -c ciphertext_file
java cs.fit.edu.rsa.RSA -d -c ciphertext_file -s secret_key_file -m plaintext_file

4. Design.

4.1. Package Diagram

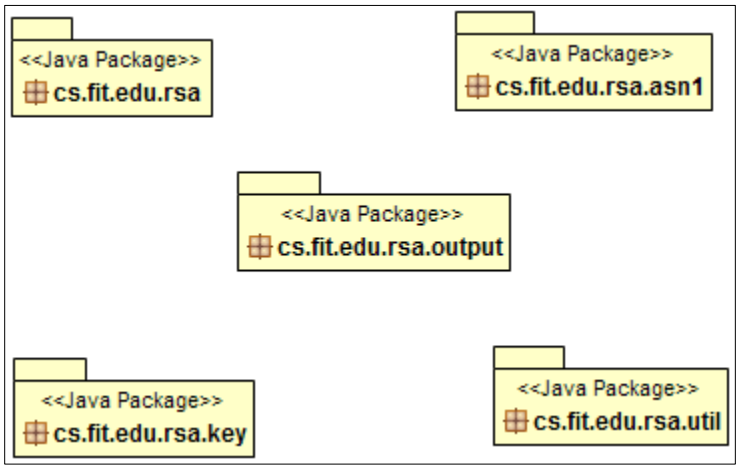

Figure 1Package Diagram for the RSA Crypto Project

The main package is the cs.fit.edu.rsa which holds the CryptoSystem classes and the Application class. The cs.fit.edu.asn1 package includes the necessary classes to handle ASN1 Encoding/Decoding using Dr. Silaghi's library.

The cs.fit.edu.rsa.key contains the RSA object models that hold the Public/Private keys. The cs.fit.edu.output classes that process console/file output. The cs.fit.edu.rsa.util holds general purpose utility classes.

4.2. Class Diagram

4.2.1. cs.fit.edu.rsa package

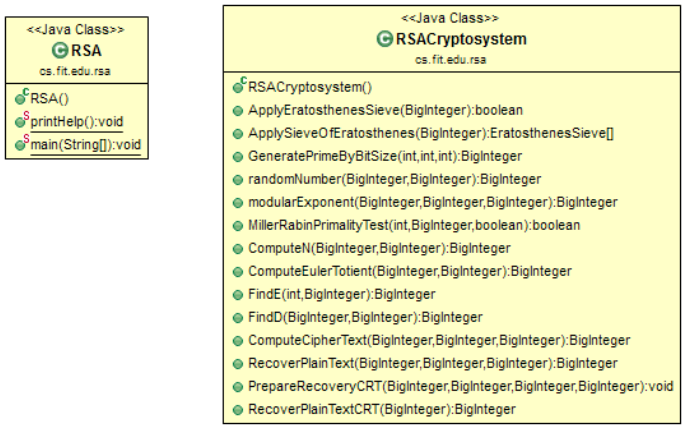

Figure 2 The cs.fit.edu.rsa package

4.2.2. cs.fit.edu.rsa.asn1 package

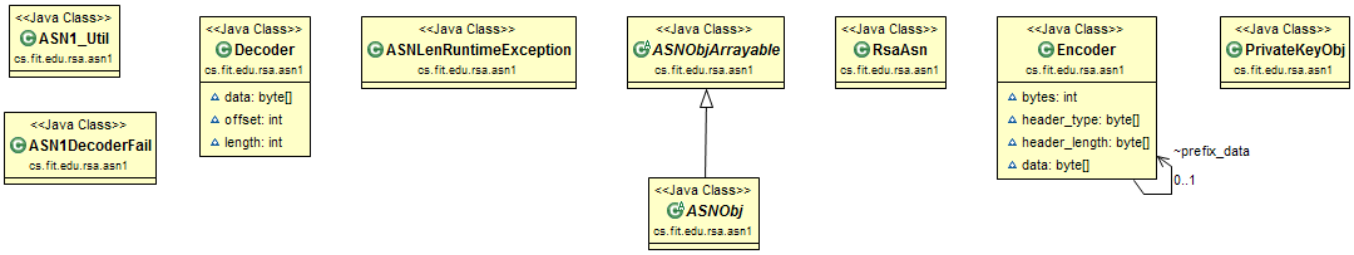

Figure 3 The class diagram for the cs.fit.edu.asn1 package

4.2.3. cs.fit.edu.rsa.output package

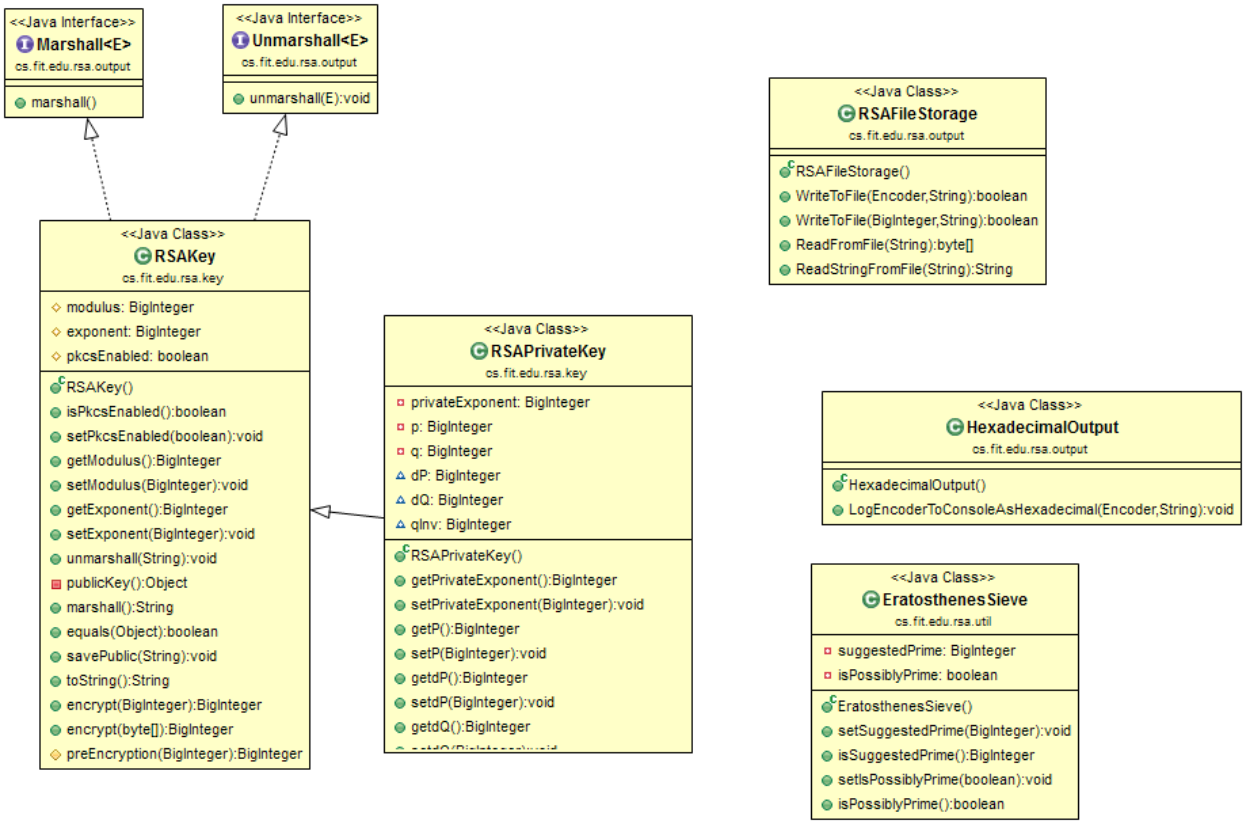

Figure 4The class diagram for the cs.fit.edu.rsa package

4.3. Design Considerations

The project was based on Object Oriented principles. The Object models RSAKey and RSAPrivateKey simply hold the attributes for the keys such as the modulus, exponent, primes and coefficient. These two classes implemented the Marshalling/Unmarshalling interfaces to serialize the model attributes for later storage. The Marshalling interface requires implementation of the *mashall()* method, respectively the Unmarshalling interface requires the unmarshall() method to recreate the model from a serialized object.

The RSACryptoSystem class implemented the Miller Rabin algorithm in the *MillerRabinPrimalityTest()* method using the modular exponentiation as well as the EratosthenesSieve. The RSAPrivateKey's *generateKey()* static methods provides an alternative way utilizing the *BigInteger.probablePrime* implementation.

The ASN1 encoding uses Dr. Silaghi's Java library and the RsaAsn encodes/decodes date to/from a byte array. The system only encodes the private/public key pair, for a future version the message will be encoded as well.

The PKCS 1.5 padding implementation is handled by RSAKey *preEncryption()* method, this option is available by setting the pkcsEnabled flag. The RSAPrivateKey *postDecryption*() method handles the unpacking of the padded bytes.

The RSA class that handles input/output from the files uses data blocks of public keys bit length encrypt/decrypt the message.

5. Test methodology:

Most of the testing focused on the encrypting/decrypting functions. Some of the unit test verify generating key pairs of different bit lengths, verifying the functions reported errors appropriately. It also included verifying the serializing/un-serializing of the keys correctly.