

Programming Assignment 1

1 Aim

The aim of this project is to familiarize you with some basic Java Crypto APIs. At the end of this project you should develop the confidence to code symmetric ciphers and hashes in Java. Other crypto API's such as the OpenSSL API (www.openssl.org) and the .NET API provide similar functionality. Knowing one API well will help you maneuver through others more easily. You should find a wealth of information on Java's implementation of crypto APIs at <http://java.sun.com/j2se/1.5.0/docs/index.html>. There's the Java API specification and the security features in J2SDK which are described at <http://java.sun.com/security/index.jsp>.

2 Problem

Implement a file encryption utility in Java that takes a file name, say *f* and encrypts its contents using AES₁₂₈ in CBC mode with PKCS5 padding. The output of the encryption should be saved in file *f.roil* if it doesn't exist. It's an error for a file named *f.roil* to exist. If one exists, you shouldn't overwrite it and terminate with an error. The format of *f.roil* should be as follows:

AES/CBC/PKCS5/MD5/ <i>n</i> \0\0\0\0\0\0\0···\0 <16 bytes of IV randomly generated>	32 bytes indicate the type of encryption for CBC mode
<32 bytes to store the hash of <i>f</i> >	encrypted
<contents of <i>f</i> >	encrypted

The shaded parts in the roil file description above indicate encrypted data. The file format starts with a description of how encrypted was performed. <AES/CBC/PKCS5/MD5/*n*> indicates that the encryption was done with the AES symmetric key block cipher in CBC mode, and that the data was padded with PKCS5 padding to accommodate for data that may not be a multiple of the block size. The integrity of the data is verified using the MD5 hash. The hash is computed over plaintext data i.e., over the contents of the original file *f* and is stored left aligned within the 32 byte space specified for storing the hash. The remaining bytes in the hash space should be padded with the NULL character '\0'.

The key for AES encryption should be derived from the MD5 hash of a passphrase input during *roiling* the file. Run MD5 *n* times on the passphrase to obtain the encryption key, where $1 \leq n < 256$. The encoding of *n* takes one byte. For e.g., if the passphrase (a password really) is "I love NUS", and *n* = 2, then you'll hash it once with MD5 to get the byte sequence 0x9fdaabb114e5908065de1e4ab49e13f9. You will hash this byte sequence again to get the actual key bytes. Use these bytes as the key for AES₁₂₈ encryption.

Your program should also be able to "unroil" a roiled file by taking a file such as *f.roil*, prompting the user for a passphrase, decrypting the file and verifying that it has the correct hash. When unroiling a file, create the file *f* with the decrypted contents. It's an error for a file named *f* to exist and you should not overwrite it if it exists. Do not write *f* if its MD5 hash does not verify, rather generate an error.

Do *not* read the whole file to be encrypted or decrypted into one **String** (or store it in memory) and operate on it. You may use temporary files to save intermediate data. To generate a random IV, use `SecureRandom`.

Your program should be in written in the *default package* and correctly build and execute on **sun-fire.comp**. The (tentative) deadline for submission is February 4.

3 Submission

You will submit a jar file that includes all the sources (.java files) and a README file that describes the contents of all other files in your submission. If you've done something extra please mention it in the README file. Submit only java source files, not any .class files. I should be able to compile your program by unjarring your submitted jar file in a directory and running `javac *.java` to create at least a `roil.class` and `unroil.class`. I should then be able to roil a file by using `java roil f` and unroil using `java unroil f.roil`.

Instructions on how to submit this file will be made available later.