Appendix M
Java Cryptographic APIs

Nick Pullman
(Citigroup Information Security division)

Supplement to
Cryptography and Network Security, Fifth Edition
William Stallings
Prentice Hall 2010
ISBN-10: 0136097049
http://williamstallings.com/Crypto/Crypto5e.html

Introduction

Many applications are currently coded using Java technologies and adoption of Java as a business enabler will continue as more companies invest in service-oriented architecture (SOA) and web services. As Java continues to find its way into the enterprise, security needs to be addressed with the design of all new applications and updated in all existing applications. Java provides an extensible security architecture with which to secure applications that are written in Java; these APIs include:

- Java Cryptography Architecture (JCA)
- Java Cryptographic Extension (JCE)
- Java Certification Path API (CertPath)
- Java Secure Socket Extension (JSSE)
- Java Authentication and Authorization Service (JAAS)
- Java Generic Secure Services (JGSS)

(Steel, Nagappan, & Lai, 2005, chap. 3)

This paper will discuss the first two APIs, JCA and JCE, in order to illustrate how cryptography can be utilized to secure information in Java. This paper will discuss the architecture and the classes of the two cryptography APIs along with examples to illustrate its uses.

JCA and JCE Architecture

The Java cryptographic APIs, JCA and JCE, both follow the same architecture for implementing and interfacing cryptographic services. In order to create an extensible framework for implementing cryptographic services, the architecture for cryptography in Java separates engine classes from service providers. JCA and JCE implement the engine classes which provide a standard interface to the service providers. The service providers are the APIs which actually implement the cryptographic algorithms and types that are acted upon. A simple way to think of this is that JCA will implement a class such as a message digest, we know what a message digest is, but just having a generic message digest doesn't tell us anything; it needs a cryptographic service. The cryptographic service provider will implement the actual algorithm, such as MD5 or SHA-1 in this example. JCA implements the generic class and the service provider implements the actual algorithm or type of cryptographic service that will be used. See figure below for an overview of the architecture along with a few example service providers listed.

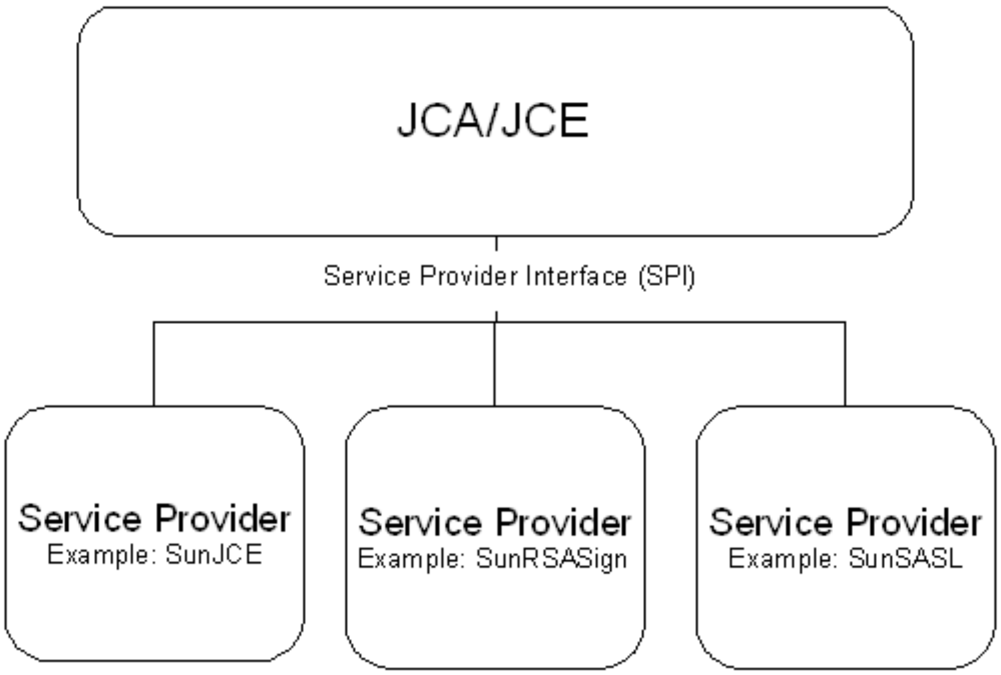


Figure 1: JCA/JCE Architecture (Steel et al., 2005, chap. 3)

This architecture is used in order to make the cryptographic APIs extensible by allowing different service providers to be "plugged in" to the framework and then utilized using standard interfaces. For example, if a new algorithm for message digests was created, the message digest class in JCA could still be utilized to implement the new algorithm. The code for instantiating a MessageDigest class in JCA looks like this:

```
MessageDigest myDigest = MessageDigest.getInstance("MD5");
```

This instantiation of the MessageDigest class uses the highest level service provider, because one wasn't specified, based on the **java.security** file which is located in **jre-home\lib\security\java.security.** The default installation of service providers from Java 1.5 should look similar to this:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=com.sun.net.ssl.internal.ssl.Provider
security.provider.4=com.sun.crypto.provider.SunJCE
security.provider.5=sun.security.jgss.SunProvider
security.provider.6=com.sun.security.sasl.Provider
```

The code for instantiating the MessageDigest class in JCA using a specific provider looks like this:

```
MessageDigest myDigest = MessageDigest.getInstance("MD5", "SUN");
```

Although the default Java 1.5 installation comes with a number of service providers, there are a number of reasons why a third party service provider would need to be installed such as vendor specific implementations or new algorithms (Kumar, 2005). Unfortunately the installation of a new service provider is somewhat convoluted, although it is for good reason. First, the service provider jar file needs to be put into a directory that is in the CLASSPATH environment variable of your operating system. Secondly, if the application is an applet or is otherwise running underneath a security manager then appropriate permissions must be granted to the application. Finally, in order for the JCE to ensure that the provider is valid, the code must be signed by a trusted Certificate Authority (CA). Currently, the two CAs that can be used are:

- **Sun Microsystems' JCE Code Signing CA**
- **IBM JCE Code Signing CA**

("How to implement a provider for the java cryptography extension")
These steps are all done in order to ensure the validity of the service provider, because if the service provider is compromised, then all of the services performed by the service provider are potentially compromised as well.

The cryptography architecture in Java was created in order to separate the actual implementation of the algorithms from the engine classes which interface with the service providers. This allows the architecture to be both flexible for the developer to decide what his/her cryptographic needs are, and also extensible so that vendors and other developers can create their own service providers to extend or implement additional functionality or algorithms. In addition, since the cryptographic services need to be very secure, Java ensures the integrity of the cryptographic algorithms by authenticating the service providers which are digitally signed by a trusted CA. This secures the architecture from attackers trying to override valid service providers with insecure and/or unverified providers. This architecture allows developers to utilize cryptographic services that are flexible, extensible, and secure.

JCA Classes

JCA along with JCE are the two main cryptographic APIs included in Java. JCA, Java Cryptographic Architecture, contains two provider classes for working with service providers and a number of engine classes used for interfacing with the cryptographic services. A table of the main classes in JCA is below.

Provider Classes	Provider	java.security.Provider
	Security	java.security.Security
Engine Classes	MessageDigest	java.security.MessageDigest
	Signature	java.security.Signature
	KeyPairGenerator	java.security.KeyPairGenerator
	KeyFactory	java.security.KeyFactory
	CertificateFactory	java.security.cert.CertificateFactory
	KeyStore	java.security.KeyStore

	AlgorithmParameters	java.security.AlgorithmParamaters
	AlgorithmParameterGenerator	java.security.algorithmParameterGenerator
	SecureRandom	java.security.SecureRandom

Table 1: Main JCA Classes (Steel et al., 2005, chap. 3)

The provider classes allow developers the ability to add, remove, and query the currently installed providers. The *Provider* class is used to query information about the installed service providers such as name and version. The *Security* class is used to add, remove, and modify the service providers. The methods of the *Security* class can only be used by a trusted application based on the Java security policy. This tries to ensure that only appropriate applications are updating, adding, and removing the service providers.

The engine classes are really the heart of JCA and this is where the interface between JCA and the actual implementation of the service classes occur. These classes are typically used by first instantiating a concrete object, which includes the service provider, and then acting upon that object. An example of this is seen below when creating a message digest. First, the data that we are going to get the message digest of is created. Secondly, the MessageDigest class, "md" in instantiated creating a concrete object using the "SHA" algorithm. Next, "md" is updated with the input data and finally the message digest is created.

```
byte[] dataBytes = "This is test data".getBytes();
MessageDigest md = MessageDigest.getInstance("SHA");
md.update(dataBytes);
byte[] digest = md.digest();
```

Below is an overview of the different engine classes and their usage.

MessageDigest: used to implement one-way hash functions such as MD5 or SHA

Signature: used to implement digital signatures

KeyPairGenerator: used to create public/private key pairs for different algorithms

KeyFactory: used to convert keys into key specifications and then vice-versa

CertificateFactory: used to generate certificates

KeyStore: used to create a keystore which maintains keys and certificates in memory for later usage

AlgorithmParameters: used to maintain the security parameters for specific algorithms

AlgorithmParameterGenerator: used to create a set of parameters to be used for specific algorithms

SecureRandom: used to create random or pseudo-random numbers

(Steel et al., 2005, chap. 3)

JCA is the API that is used to perform many operations used in cryptography such as digital signatures, certificate generation, public/private key generation, and message digests. JCA also has a set of classes that is used to manage the service providers that are used to implement the cryptographic algorithms and types. Using the default service provider "SUN" coupled with the JCA API, Java provides a solid architecture to perform many cryptographic functions. In the next section, the paper will discuss the JCE API which is used to perform many of the cryptographic functions that JCA does not implement.

JCE Classes

Although JCA provides a number of cryptographic functions in Java, JCE provides many of the functions that JCA does not provide such as the actual encryption/decryption and symmetric key generation. JCE uses the same architecture as JCA with engine classes which are abstract and service providers which implement the algorithms. JCE, unlike JCA, does not include any provider classes itself and instead utilizes the JCA classes "Provider" and "Security" to add, remove, and query the service providers which means that JCE is made up entirely of engine classes. A table of the main classes of JCE is below.

Engine Classes	Cipher	javax.crypto.Cipher
	CipherInputStream	javax.crypto.CipherInputStream
	CipherOutputStream	javax.crypto.CipherOutputStream
	Mac	javax.crypto.Mac
	KeyGenerator	javax.crypto.KeyGenerator
	SecretKeyFactory	javax.crypto.SecretKeyFactory
	SealedObject	javax.crypto.SealedObject
	KeyAgreement	javax.crypto.KeyAgreement
	interfaces	javax.crypto.interfaces
	spec	javax.crypto.spec

Table 2: Main JCE Classes (Steel et al., 2005, chap. 3)

Since JCE uses the same architecture as JCA, the code is quite similar. For example, in order to generate a DES key, first a concrete **KeyGenerator** object is created using the **getInstance()** method with "DES" as a parameter. Then a secret key is generated using the KeyGenerator's **generateKey()** method. The code to create a secret DES key is show below.

```
KeyGenerator keyGen = KeyGenerator.getInstance("DES");
SecretKey key = keygens.generateKey();
```

Notice that this code is quite similar to creating a message digest because the overall architecture is the same.

Below is an overview of the engine classes in JCE and their subsequent uses.

Cipher: provides encryption and decryption functionality

CipherInputStream & CipherOutputStream: used as a convenient way to encrypt or decrypt information using streams

Mac: used to check the integrity of messages based on a secret key

KeyGenerator: used to generate symmetric keys

SecretKeyFactory: similar to the KeyFactory of JCA which converts keys into key specifications and vice-versa

SealedObject: used to create a serialized object which is protected using cryptography

KeyAgreement: provides functionality to use a key agreement protocol

Interfaces: provides interfaces for Diffie-Hellman keys

Spec: similar to algorithmParamaters of JCA which provides key and parameter specifications for different algorithms

(Steel et al., 2005, chap. 3)

JCE, in conjunction with the default cryptographic service providers, provide a solution to take care of most cryptographic needs that aren't performed by JCA. If additional functionality is needed beyond the default providers, new providers can be "plugged in" to the JCE architecture to provide the functionality. The Java implementation of cryptographic functions is both robust and flexible. In addition, the java cryptographic functions are more than satisfactory for most implementations. Overall, JCE provides the most typical functionality that is needed for cryptographic processing.

Conclusion

Java continues to be used more and more for enterprise applications, especially since there has been a push toward more web services and SOA. As more enterprises adopt Java as a core technology, it becomes critical for it to securely handle storage and transmission. Java has created a number of APIs that meet the need for cryptographic functionality. The core of Java cryptography is JCA and JCE which work together in conjunction with cryptographic service providers to provide cryptographic functionality. The architecture of JCA/JCE was setup to be flexible by allowing different service providers to be "plugged in" to the JCA/JCE architecture. In the end, most cryptographic needs will be taken care of by JCA/JCE and the default cryptographic providers and there are commercial and open source service providers available in case additional functionality is needed.

Works Cited

Class keystore. Available: http://java.sun.com/j2se/1.5.0/docs/api/java/security/KeyStore.html#deleteEntry(java.lang.String)

How to implement a provider for the java cryptography architecture (January 2003). Available: http://java.sun.com/j2se/1.4.2/docs/guide/security/HowToImplAProvider.html

How to implement a provider for the java cryptography extension. Available: http://java.sun.com/j2se/1.5.0/docs/guide/security/jce/HowToImplAJCEProvider.html#Step%205a

Java cryptographic architecture API specification and reference (July 2004). Available: http://java.sun.com/j2se/1.5.0/docs/guide/security/CryptoSpec.html

Java cryptographic extension (JCE) reference guide (January 2004). Available: http://java.sun.com/j2se/1.5.0/docs/guide/security/jce/JCERefGuide.html

Java security for the enterprise. Available: http://www.j2ee-security.net/book/dnldsrc/

Kumar, P (May 2004). J2EE security for servlets, EJBs, and web services. Prentice Hall PTR. Available: http://www.informit.com/articles/article.asp?p=170967&seqNum=0

Steel, C & Nagappan, R & Lai, R (September 2005). Core security patters: best practices and strategies for J2EE, web services, and identity management. Prentice Hall PTR.

Appendix A: Using the Cryptographic Application

There are three main functions that this application provides: creating and comparing message digests, creating and maintaining key stores, and finally encrypting and decrypting files. These three main operations are detailed below.

Message Digests

First, in order to perform any operations, we need to open up a file to work with. Click ***File: Open*** from the menu bar, and find a file to work with.

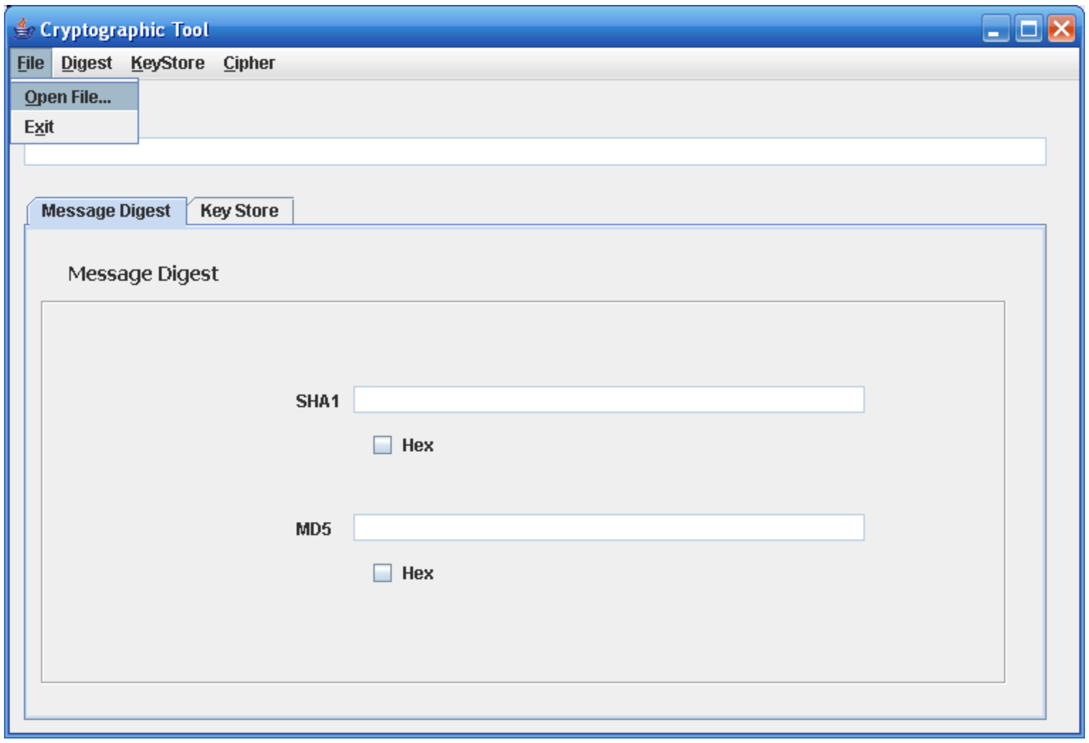


Figure 2: Open File

Then, click on ***Digest: Get Digest: MD5 & SHA1*** in order to get the message digests. The message digests can then be saved by clicking ***Digest: Save Digest*** and then clicking the digest that you want to save. A popup will then ask you where to save it. The filename will default to the input filename with either SHA or MD5 as the extension.

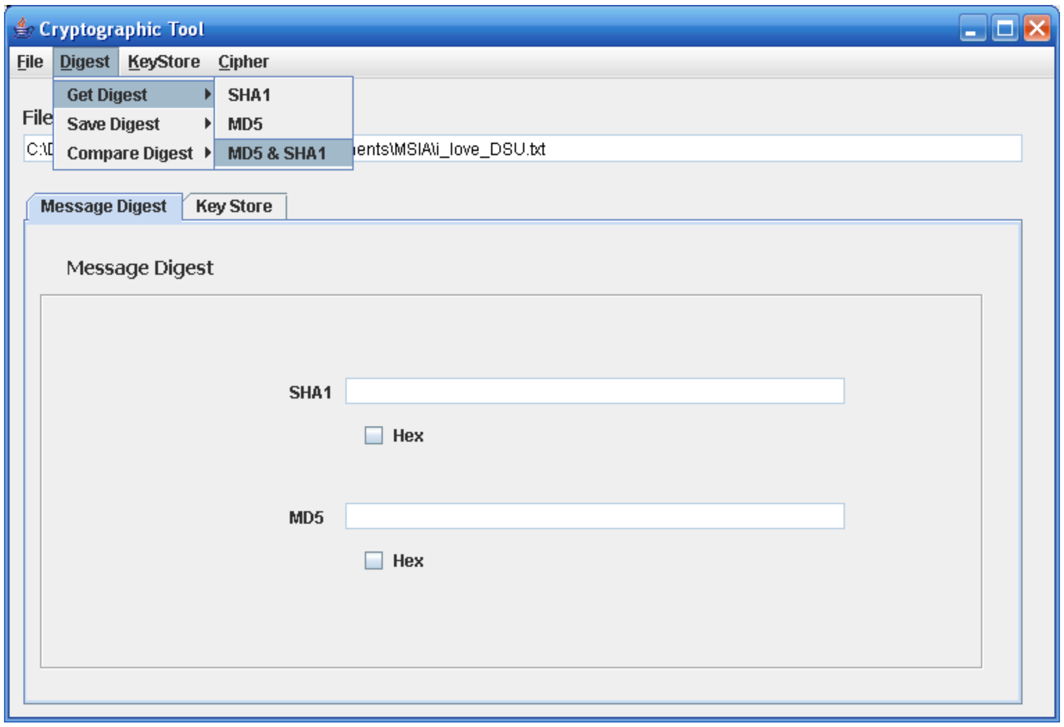
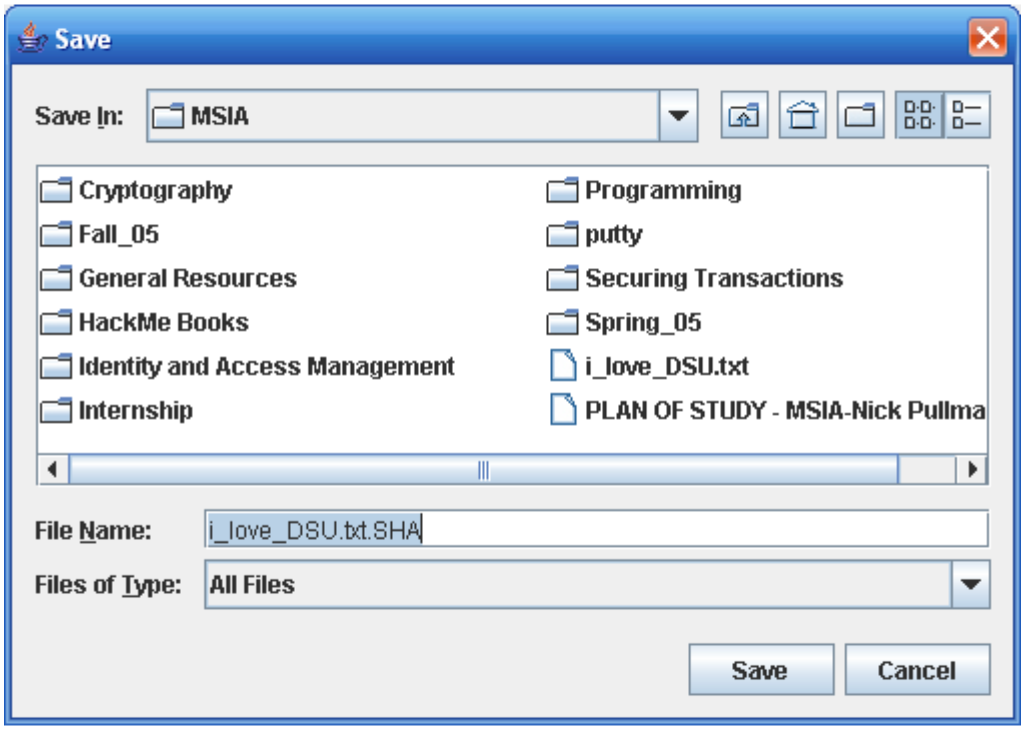


Figure 3: Get Digests



File Name: i_love_DSU.txt.SHA

Files of Type: All Files

Figure 4: Save Digest

Then, if you want to compare the current file's digest to a digest that was previously saved to ensure that they are equal, you can click on ***Digest: Compute Digest*** and then choose the correct digest, which will pop up a file chooser. Then, pick the file that contains the message digest that you want to compare to. A message box will alert you as to whether the message digests matched or did not match.

Key Stores

To create a key store, click on ***KeyStore: Create Keystore***. Choose a location and filename from the file chooser that pops up, and the input a password in order to protect the key store.

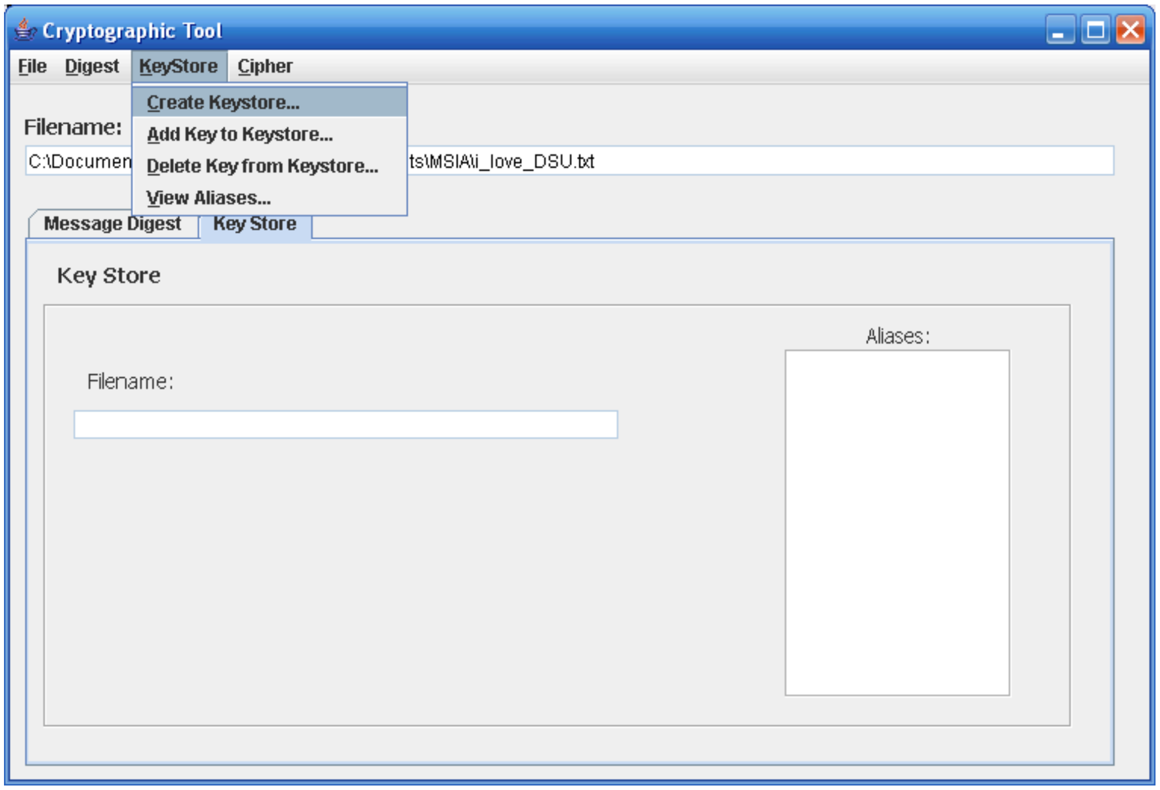
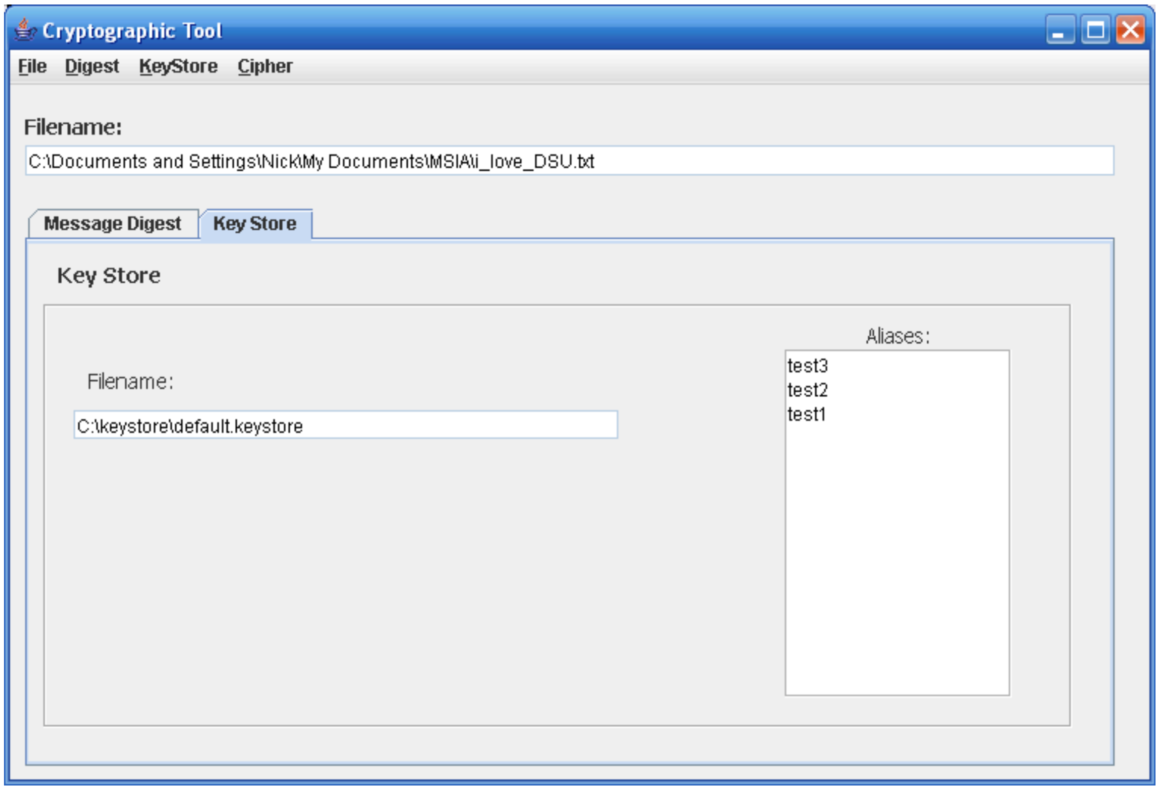


Figure 5: Create Key Store

This creates an empty key store, now you have to add keys to the key store. This is done by clicking ***KeyStore: Add Key to Keystore***. Then, choose the key store that you want to add the key to, on the input box type in the password and click ***OK***, finally type in the alias of the key in order to distinguish it from different keys. Removing keys is almost identical, except instead of the key being added, the key will be removed from the key store. The View Aliases option allows you to query the key store in order to display all of the aliases, the keys, which are in the key store. The list shows up in the text box on the key store tab.



Filename:

C:\Documents and Settings\Nick\My Documents\MSIA\i_love_DSU.txt

Key Store

Filename:

C:\keystore\default.keystore

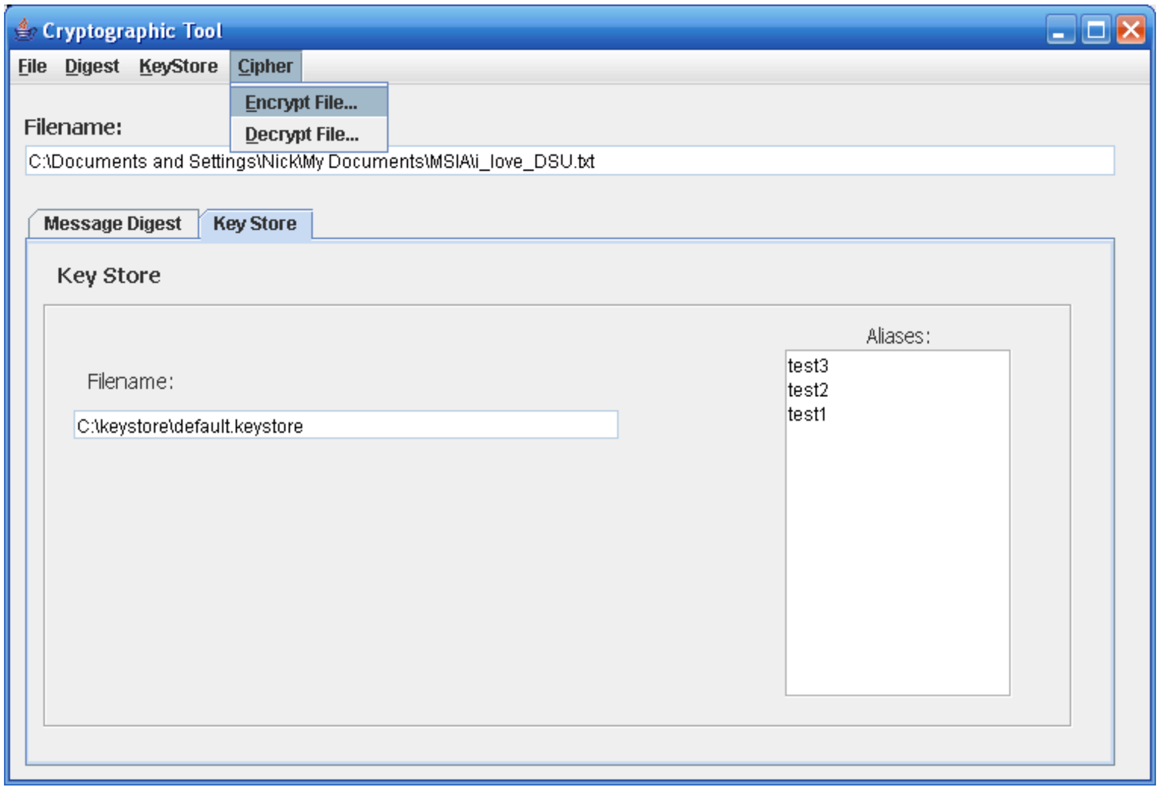
Aliases:

test3
test2
test1

Figure 6: Key Store tab showing aliases

Encrypting/Decrypting

Encryption/Decryption is done by choosing a key from a key store and choosing where to save the file, where the key store is, the password, and the alias. This will create the encrypted file. Decrypting the file is very similar by selecting the encrypted file, selecting the output of the decrypted file, selecting the key store, typing the password, and then typing the alias of the key to decrypt the file.



Appendix B: JCA/JCE Cryptography Example

In order to provide some examples of how JCA/JCE can be used to provide cryptographic functionality, I wrote a small application that allows the user to create and compare message digests, create a keystore, and encrypt/decrypt a file using DES. The java code, an embedded jar executable, and an embedded zip file which is the project folder that can be opened with Netbeans are below. The code below is original, but derived from examples seen from the following references: (Steel et al., 2005, chap. 3), ("Java security for the enterprise"), and (Kumar, May 2004).

Executable JAR

C:\Documents and
Settings\Nick\My Docu

Zipped Project Folder

C:\Documents and
Settings\Nick\My Docu

Source Code for Application

****

* Source Code for CryptMain

****

```
/*
 * mainFrame.java
 *
 * Created on January 21, 2006, 4:41 PM
 * Cryptography tool 1.0
 */

package crypt;
import java.io.File;
import java.io.*;
import java.security.MessageDigest;
import java.util.Enumeration;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;



/**
 *
 * @author  Nick Pullman
 */
public class CryptMain extends javax.swing.JFrame {
    String pathname = new String();
//pathname of file that we will be working on
    String MD5Digest = new String();                                        //Digest
of file using MD5 hash
    String SHA1Digest = new String();                                       //Digest
of file using SHA1
    String keyFilePath = new String();
//Absolute pathname to keystore
    String password = new String();
//Password to be used for keystore

    /** Creates new form mainFrame */
    public CryptMain() {
        initComponents();
            }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
    private void initComponents() {
        txtInFilePathName = new javax.swing.JTextField();
        jLabel3 = new javax.swing.JLabel();
        jTabbedPane1 = new javax.swing.JTabbedPane();
        jPanel1 = new javax.swing.JPanel();
        jPanel4 = new javax.swing.JPanel();
        chkHexMD5 = new javax.swing.JCheckBox();
        txtMD5 = new javax.swing.JTextField();
        jLabel2 = new javax.swing.JLabel();
        chkHexSHA1 = new javax.swing.JCheckBox();
        txtSHA1 = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jPanel2 = new javax.swing.JPanel();
        jPanel3 = new javax.swing.JPanel();
        jLabel5 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
```

```
txtKeyStorePath = new javax.swing.JTextField();
txtAliases = new javax.swing.JTextArea();
jLabel4 = new javax.swing.JLabel();
jMenuBar1 = new javax.swing.JMenuBar();
jMenu1 = new javax.swing.JMenu();
menuItemOpenFile = new javax.swing.JMenuItem();
menuItemExit = new javax.swing.JMenuItem();
jMenu2 = new javax.swing.JMenu();
jMenu3 = new javax.swing.JMenu();
menuItemGetSHA1 = new javax.swing.JMenuItem();
menuItemGetMD5 = new javax.swing.JMenuItem();
menuItemGetBoth = new javax.swing.JMenuItem();
jMenu4 = new javax.swing.JMenu();
menuItemSaveSHA1 = new javax.swing.JMenuItem();
menuItemSaveMD5 = new javax.swing.JMenuItem();
jMenu5 = new javax.swing.JMenu();
menuItemCompSHA1 = new javax.swing.JMenuItem();
menuItemCompMD5 = new javax.swing.JMenuItem();
jMenu6 = new javax.swing.JMenu();
menuItemCreateKeyStore = new javax.swing.JMenuItem();
menuItemAddKey = new javax.swing.JMenuItem();
menuItemDeleteKey = new javax.swing.JMenuItem();
menuItemViewAliases = new javax.swing.JMenuItem();
jMenu7 = new javax.swing.JMenu();
menuItemEncryptFile = new javax.swing.JMenuItem();
menuItemDecryptFile = new javax.swing.JMenuItem();

getContentPane().setLayout(null);

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setTitle("Cryptographic Tool");
setName("frame");
txtInFilePathName.setDisabledTextColor(new java.awt.Color(0, 0, 0));
txtInFilePathName.setEnabled(false);
getContentPane().add(txtInFilePathName);
txtInFilePathName.setBounds(10, 40, 720, 20);

jLabel3.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel3.setText("Filename:");
getContentPane().add(jLabel3);
jLabel3.setBounds(10, 20, 100, 14);

jPanel1.setLayout(null);

jPanel4.setLayout(null);

jPanel4.setBorder(new javax.swing.border.EtchedBorder());
chkHexMD5.setText("Hex");
chkHexMD5.setIconTextGap(8);
chkHexMD5.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        chkHexMD5ActionPerformed(evt);
    }
});

jPanel4.add(chkHexMD5);
chkHexMD5.setBounds(230, 180, 60, 23);

txtMD5.setDisabledTextColor(new java.awt.Color(0, 0, 0));
txtMD5.setEnabled(false);
jPanel4.add(txtMD5);
txtMD5.setBounds(220, 150, 360, 19);

jLabel2.setText("MD5");
jPanel4.add(jLabel2);
jLabel2.setBounds(180, 150, 30, 20);

chkHexSHA1.setText("Hex");
chkHexSHA1.setIconTextGap(8);
chkHexSHA1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        chkHexSHA1ActionPerformed(evt);
    }
});

jPanel4.add(chkHexSHA1);
chkHexSHA1.setBounds(230, 90, 60, 23);

txtSHA1.setAutoscrolls(false);
txtSHA1.setDisabledTextColor(new java.awt.Color(0, 0, 0));
txtSHA1.setEnabled(false);
jPanel4.add(txtSHA1);
txtSHA1.setBounds(220, 60, 360, 19);

jLabel1.setText("SHA1");
jPanel4.add(jLabel1);
jLabel1.setBounds(180, 60, 40, 20);

jPanel1.add(jPanel4);
jPanel4.setBounds(10, 50, 680, 270);

jLabel7.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel7.setText("Message Digest");
jPanel1.add(jLabel7);
jLabel7.setBounds(30, 20, 120, 20);

jTabbedPane1.addTab("Message Digest", jPanel1);

jPanel2.setLayout(null);

jPanel3.setLayout(null);

jPanel3.setBorder(new javax.swing.border.EtchedBorder());
jLabel5.setFont(new java.awt.Font("Tahoma", 0, 14));
jLabel5.setText("Filename:");
jPanel3.add(jLabel5);
jLabel5.setBounds(30, 40, 110, 20);

jLabel6.setFont(new java.awt.Font("Tahoma", 0, 14));
jLabel6.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel6.setText("Aliases:");
jPanel3.add(jLabel6);
jLabel6.setBounds(490, 10, 150, 20);

txtKeyStorePath.setDisabledTextColor(new java.awt.Color(0, 0, 0));
txtKeyStorePath.setEnabled(false);
jPanel3.add(txtKeyStorePath);
txtKeyStorePath.setBounds(20, 70, 360, 19);

txtAliases.setBorder(new javax.swing.border.EtchedBorder());
txtAliases.setDisabledTextColor(new java.awt.Color(0, 0, 0));
txtAliases.setEnabled(false);
jPanel3.add(txtAliases);
txtAliases.setBounds(490, 30, 150, 230);

jPanel2.add(jPanel3);
jPanel3.setBounds(10, 40, 680, 280);

jLabel4.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel4.setText("Key Store");
jPanel2.add(jLabel4);
jLabel4.setBounds(20, 10, 110, 20);

jTabbedPane1.addTab("Key Store", jPanel2);

getContentPane().add(jTabbedPane1);
jTabbedPane1.setBounds(10, 80, 720, 370);

jMenu1.setMnemonic('f');
jMenu1.setText("File");
menuItemOpenFile.setMnemonic('o');
menuItemOpenFile.setText("Open File...");
```

```
menuItemOpenFile.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        menuItemOpenFileActionPerformed(evt);
    }
});

jMenu1.add(menuItemOpenFile);

menuItemExit.setMnemonic('x');
menuItemExit.setText("Exit");
menuItemExit.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        menuItemExitActionPerformed(evt);
    }
});

jMenu1.add(menuItemExit);

jMenuBar1.add(jMenu1);

jMenu2.setMnemonic('d');
jMenu2.setText("Digest");
jMenu3.setText("Get Digest");
menuItemGetSHA1.setText("SHA1");
menuItemGetSHA1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        menuItemGetSHA1ActionPerformed(evt);
    }
});

jMenu3.add(menuItemGetSHA1);

menuItemGetMD5.setText("MD5");
menuItemGetMD5.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        menuItemGetMD5ActionPerformed(evt);
    }
});

jMenu3.add(menuItemGetMD5);

menuItemGetBoth.setText("MD5 & SHA1");
menuItemGetBoth.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        menuItemGetBothActionPerformed(evt);
    }
});

jMenu3.add(menuItemGetBoth);

jMenu2.add(jMenu3);

jMenu4.setText("Save Digest");
menuItemSaveSHA1.setText("Save SHA1...");
menuItemSaveSHA1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        menuItemSaveSHA1ActionPerformed(evt);
    }
});

jMenu4.add(menuItemSaveSHA1);

menuItemSaveMD5.setText("Save MD5...");
menuItemSaveMD5.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        menuItemSaveMD5ActionPerformed(evt);
    }
});

jMenu4.add(menuItemSaveMD5);
```

```
jMenu2.add(jMenu4);

jMenu5.setText("Compare Digest");
menuItemCompSHA1.setText("SHA1 to File...");
menuItemCompSHA1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        menuItemCompSHA1ActionPerformed(evt);
    }
});

jMenu5.add(menuItemCompSHA1);

menuItemCompMD5.setText("MD5 to File...");
menuItemCompMD5.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        menuItemCompMD5ActionPerformed(evt);
    }
});

jMenu5.add(menuItemCompMD5);

jMenu2.add(jMenu5);

jMenuBar1.add(jMenu2);

jMenu6.setMnemonic('k');
jMenu6.setText("KeyStore");
menuItemCreateKeyStore.setMnemonic('c');
menuItemCreateKeyStore.setText("Create Keystore...");
menuItemCreateKeyStore.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        menuItemCreateKeyStoreActionPerformed(evt);
    }
});

jMenu6.add(menuItemCreateKeyStore);

menuItemAddKey.setMnemonic('a');
menuItemAddKey.setText("Add Key to Keystore...");
menuItemAddKey.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        menuItemAddKeyActionPerformed(evt);
    }
});

jMenu6.add(menuItemAddKey);

menuItemDeleteKey.setMnemonic('d');
menuItemDeleteKey.setText("Delete Key from Keystore...");
menuItemDeleteKey.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        menuItemDeleteKeyActionPerformed(evt);
    }
});

jMenu6.add(menuItemDeleteKey);

menuItemViewAliases.setMnemonic('v');
menuItemViewAliases.setText("View Aliases...");
menuItemViewAliases.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        menuItemViewAliasesActionPerformed(evt);
    }
});

jMenu6.add(menuItemViewAliases);

jMenuBar1.add(jMenu6);

jMenu7.setMnemonic('c');
jMenu7.setText("Cipher");
```

```
        menuItemEncryptFile.setMnemonic('e');
        menuItemEncryptFile.setText("Encrypt File...");
        menuItemEncryptFile.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                menuItemEncryptFileActionPerformed(evt);
            }
        });

        jMenu7.add(menuItemEncryptFile);

        menuItemDecryptFile.setMnemonic('d');
        menuItemDecryptFile.setText("Decrypt File...");
        menuItemDecryptFile.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                menuItemDecryptFileActionPerformed(evt);
            }
        });

        jMenu7.add(menuItemDecryptFile);

        jMenuBar1.add(jMenu7);

        setJMenuBar(jMenuBar1);

        java.awt.Dimension screenSize =
java.awt.Toolkit.getDefaultToolkit().getScreenSize();
        setBounds((screenSize.width-761)/2, (screenSize.height-516)/2, 761, 516);
    }
    // </editor-fold>

    /**Delete a key from a keystore */
    private void menuItemDeleteKeyActionPerformed(java.awt.event.ActionEvent evt) {
        String keyStorePath = new String();                                     //key
store path with local scope (i.e. this method)
        KeyGenerator kg = null;
        SecretKey sKey = null;
        JFileChooser chooser = new JFileChooser();
        File defaultFile = new File("default.keystore");
        chooser.setSelectedFile(defaultFile);
        int returnVal = chooser.showOpenDialog(null);

        if (returnVal == chooser.CANCEL_OPTION){                                //exit
method if user selects "Cancel"
            return;
        }
        keyStorePath = chooser.getSelectedFile().getAbsolutePath();             //set
pathname for file to work with
        txtKeyStorePath.setText(keyFilePath);                                   //set key
store text box to key path

        password = JOptionPane.showInputDialog("Please input the keystore password");
        String alias = JOptionPane.showInputDialog("Please input an alias for the key
that you want to delete");
        CryptKeyStore keyStore = new CryptKeyStore();


        int result = keyStore.deleteKey(keyStorePath, password, alias);

        if (result == 1){
            JOptionPane.showMessageDialog(null,"Key Successfully
Deleted","Keystore",JOptionPane.INFORMATION_MESSAGE);
            //Now update the aliases text box to show the new set of aliases.
            Enumeration aliases = keyStore.getAliases(keyStorePath, password);
            StringBuffer strAliases = new StringBuffer();
            for (int i = 0; aliases.hasMoreElements(); i++){
                strAliases.append(aliases.nextElement() + "\n");
//Load StringBuffer with all aliases of the given KeyStore
            }
            String kStoreAliases = strAliases.toString();
            txtAliases.setText(kStoreAliases);
        }else{
            JOptionPane.showMessageDialog(null,"Key Deletion
Failed","Keystore",JOptionPane.WARNING_MESSAGE);
        }
    }

    /** Get the aliases of the keystore */
    private void menuItemViewAliasesActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser chooser = new JFileChooser();
        File defaultFile = new File("default.keystore");
        chooser.setSelectedFile(defaultFile);
        int returnVal = chooser.showOpenDialog(null);

        if (returnVal == chooser.CANCEL_OPTION){   //exit method if user selects "Cancel"
            return;
        }
        keyFilePath = chooser.getSelectedFile().getAbsolutePath();              //set
pathname for file to work with
        txtKeyStorePath.setText(keyFilePath);                                   //set key
store text box to key path

        password = JOptionPane.showInputDialog("Please input a password");
        CryptKeyStore keyStore = new CryptKeyStore();
        Enumeration aliases = keyStore.getAliases(keyFilePath, password);

        StringBuffer strAliases = new StringBuffer();
        for (int i = 0; aliases.hasMoreElements(); i++){
            strAliases.append(aliases.nextElement() + "\n");
//Load StringBuffer with all aliases of the given KeyStore
        }

        String kStoreAliases = strAliases.toString();
        txtAliases.setText(kStoreAliases);
    }

    /** This method gets a secret key from a keystore and encrypts a file*/
    private void menuItemEncryptFileActionPerformed(java.awt.event.ActionEvent evt) {
//******************************************
//** Get pathname to save the encrypted file
//******************************************
        JFileChooser chooser = new JFileChooser();
        File DESDir = new File(pathname);
        File DESOutputFileTemp = new File(pathname, DESDir.getName() + ".ENCRYPTED");
        chooser.setCurrentDirectory(DESDir);
        chooser.setSelectedFile(DESOutputFileTemp);
        int returnVal = chooser.showSaveDialog(null);
        File DESOutputFile = new File(chooser.getSelectedFile().getAbsolutePath());
        String outFile = chooser.getSelectedFile().getAbsolutePath();

        //Check to see if file exists and if it does ask user if he/she indeed wants to
overwrite file
        if (DESOutputFile.isFile()){
            if(JOptionPane.showConfirmDialog(null, "File Exists. Overwrite File?",
"Confim", JOptionPane.YES_NO_OPTION) != JOptionPane.YES_OPTION){
                this.menuItemSaveSHA1ActionPerformed(evt);
                return;
            }
        }

        if (returnVal == chooser.CANCEL_OPTION){
            return;
        }

//**********************************************
//* Get pathname, alias, and password of the KeyStore File
//**********************************************
        JFileChooser keyStoreChooser = new JFileChooser();
        File defaultFile = new File("default.keystore");
        keyStoreChooser.setSelectedFile(defaultFile);
        int returnVal2 = keyStoreChooser.showOpenDialog(null);
        if (returnVal2 == keyStoreChooser.CANCEL_OPTION){                        //exit
method if user selects "Cancel"
            return;
        }
        String keyStorePath = keyStoreChooser.getSelectedFile().getAbsolutePath();
//set pathname for file to work with
        txtKeyStorePath.setText(keyFilePath);                                   //set key
store text box to key path

        password = JOptionPane.showInputDialog("Please input the keystore password");
        String alias = JOptionPane.showInputDialog("Please input an alias for this key");
//*********************************************
//* Get secret key from keystore using keyStorePath
//*********************************************
        CryptKeyStore keyStore = new CryptKeyStore();
        SecretKey sKey = keyStore.getKey(keyStorePath, password, alias);

        CryptCipher cipher = new CryptCipher();
        int result = cipher.encrypt(pathname, outFile, sKey);

        if (result == 1){
            JOptionPane.showMessageDialog(null,"File Successfully
Encrypted","Cipher",JOptionPane.INFORMATION_MESSAGE);
        }else{
            JOptionPane.showMessageDialog(null,"File Encryption was not
Successful","Cipher",JOptionPane.WARNING_MESSAGE);
        }
    }

    /** This method gets a secret key from a keystore and decrypts a file*/
    private void menuItemDecryptFileActionPerformed(java.awt.event.ActionEvent evt) {
//*******************************************
//** Get pathname of the encrypted input file
//*******************************************
        JFileChooser encryptedChooser = new JFileChooser();
        int returnVal0 = encryptedChooser.showOpenDialog(null);

        if (returnVal0 == encryptedChooser.CANCEL_OPTION){                      //exit
method if user selects "Cancel"
            return;
        }
        String encryptedPath = encryptedChooser.getSelectedFile().getAbsolutePath();
//set pathname for file to work with

//*******************************************
//** Get pathname to save the decrypted file
//*******************************************
        JFileChooser chooser = new JFileChooser();

        File DESDir = new File(encryptedChooser.getCurrentDirectory().getAbsolutePath());
        File DESOutputFileTemp = new
File(encryptedChooser.getCurrentDirectory().getAbsolutePath(), encryptedPath.substring(0,
encryptedPath.lastIndexOf(".")) + ".DECRYPTED");
        chooser.setCurrentDirectory(DESDir);
        chooser.setSelectedFile(DESOutputFileTemp);

        int returnVal = chooser.showSaveDialog(null);
        File DESOutputFile = new File(chooser.getSelectedFile().getAbsolutePath());
        String outFile = chooser.getSelectedFile().getAbsolutePath();

        //Check to see if file exists and if it does ask user if he/she indeed wants to
overwrite file
        if (DESOutputFile.isFile()){
            if(JOptionPane.showConfirmDialog(null, "File Exists. Overwrite File?",
"Confim", JOptionPane.YES_NO_OPTION) != JOptionPane.YES_OPTION){
                this.menuItemSaveSHA1ActionPerformed(evt);
                return;
            }
        }

        if (returnVal == chooser.CANCEL_OPTION){
            return;
        }

//***********************************************
//* Get pathname, alias, and password of the KeyStore File
//***********************************************
        JFileChooser keyStoreChooser = new JFileChooser();
        File defaultFile = new File("default.keystore");
        keyStoreChooser.setSelectedFile(defaultFile);
        int returnVal2 = keyStoreChooser.showOpenDialog(null);

        if (returnVal2 == keyStoreChooser.CANCEL_OPTION){                        //exit
method if user selects "Cancel"
            return;
        }
        String keyStorePath = keyStoreChooser.getSelectedFile().getAbsolutePath();
//set pathname for file to work with
        txtKeyStorePath.setText(keyFilePath);                                   //set key
store text box to key path

        password = JOptionPane.showInputDialog("Please input the keystore password");
        String alias = JOptionPane.showInputDialog("Please input an alias for this key");
//***********************************************
//* Get secret key from keystore using keyStorePath
//***********************************************
        CryptKeyStore keyStore = new CryptKeyStore();
        SecretKey sKey = keyStore.getKey(keyStorePath, password, alias);

        CryptCipher cipher = new CryptCipher();
        int result = cipher.decrypt(encryptedPath, outFile, sKey);

        if (result == 1){
            JOptionPane.showMessageDialog(null,"File Successfully
Decrypted","Cipher",JOptionPane.INFORMATION_MESSAGE);
        }else{
            JOptionPane.showMessageDialog(null,"File Decryption was not
Successful","Cipher",JOptionPane.WARNING_MESSAGE);
        }
    }

    /** This method adds an item to a keystore*/
    private void menuItemAddKeyActionPerformed(java.awt.event.ActionEvent evt) {
        String keyStorePath = new String();                                     //key
store path with local scope (i.e. this method)
        KeyGenerator kg = null;
        SecretKey sKey = null;
        JFileChooser chooser = new JFileChooser();
        File defaultFile = new File("default.keystore");
        chooser.setSelectedFile(defaultFile);
        int returnVal = chooser.showOpenDialog(null);

        if (returnVal == chooser.CANCEL_OPTION){                                //exit
method if user selects "Cancel"
            return;
        }
        keyStorePath = chooser.getSelectedFile().getAbsolutePath();             //set
pathname for file to work with
        txtKeyStorePath.setText(keyFilePath);                                   //set key
store text box to key path

        password = JOptionPane.showInputDialog("Please input the keystore password");
        String alias = JOptionPane.showInputDialog("Please input an alias for this key");
        CryptKeyStore keyStore = new CryptKeyStore();

        try {
            kg = KeyGenerator.getInstance("DES");
            sKey = kg.generateKey();
        }catch (Exception e){
            System.out.println(e);
        }
        int result = keyStore.addKey(keyStorePath, password, sKey, alias);

        if (result == 1){
            JOptionPane.showMessageDialog(null,"Key Successfully
Added","Keystore",JOptionPane.INFORMATION_MESSAGE);
            Enumeration aliases = keyStore.getAliases(keyStorePath, password);
            StringBuffer strAliases = new StringBuffer();
            for (int i = 0; aliases.hasMoreElements(); i++){
                strAliases.append(aliases.nextElement() + "\n");
//Load StringBuffer with all aliases of the given KeyStore
            }
            String kStoreAliases = strAliases.toString();
            txtAliases.setText(kStoreAliases);
        }else{
            JOptionPane.showMessageDialog(null,"Key Addition
Failed","Keystore",JOptionPane.WARNING_MESSAGE);
        }
    }

    /** This method creates a keystore */
    private void menuItemCreateKeyStoreActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser chooser = new JFileChooser();
        File defaultFile = new File("default.keystore");
        chooser.setSelectedFile(defaultFile);
        int returnVal = chooser.showOpenDialog(null);

        if (returnVal == chooser.CANCEL_OPTION){   //exit method if user selects "Cancel"
            return;
        }
        keyFilePath = chooser.getSelectedFile().getAbsolutePath();              //set
pathname for file to work with
        txtKeyStorePath.setText(keyFilePath);                                   //set key
store text box to key path

        password = JOptionPane.showInputDialog("Please input a password");
        CryptKeyStore keyStore = new CryptKeyStore();
        int result = keyStore.createKeystore(keyFilePath, password);

        if (result == 1){
            JOptionPane.showMessageDialog(null,"Key Store Successfully
Created","Keystore",JOptionPane.INFORMATION_MESSAGE);
        }else{
            JOptionPane.showMessageDialog(null,"Key Store Creation
Failed","Keystore",JOptionPane.WARNING_MESSAGE);
        }
    }

    /** This method fires both events to get the MD5 digest and the SHA1 digest */
    private void menuItemGetBothActionPerformed(java.awt.event.ActionEvent evt) {
        this.menuItemGetMD5ActionPerformed(evt);   //Get MD5 Digest
        this.menuItemGetSHA1ActionPerformed(evt);  //Get SHA1 Digest
    }

   /**This method compares the current MD5 digest with a SHA1 digest
     * that is located in a file */
    private void menuItemCompMD5ActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser chooser = new JFileChooser();
        File MD5Dir = new File(pathname);
        File MD5OutputFileTemp = new File(pathname, MD5Dir.getName() + ".MD5");
        chooser.setCurrentDirectory(MD5Dir);
        chooser.setSelectedFile(MD5OutputFileTemp);
        int returnVal = chooser.showOpenDialog(null);

        if (returnVal == chooser.CANCEL_OPTION){
            return;
        }

        StringBuffer contents = new StringBuffer();
        BufferedReader input = null;
        try {
            input = new BufferedReader( new
FileReader(chooser.getSelectedFile().getAbsolutePath()));
            String line = null; //not declared within while loop

            while (( line = input.readLine()) != null){
                contents.append(line);
            }
        }catch (FileNotFoundException ex) {
            ex.printStackTrace();
        }catch (IOException ex){
              ex.printStackTrace();
        }
        finally {
            try {
                if (input!= null) {
                //flush and close both "input" and its underlying FileReader
                input.close();
                }
            }catch (IOException ex) {
                    ex.printStackTrace();
            }
        }

        MessageDigest md = null;
        boolean equalDigest;
        try{
            MessageDigest.getInstance("MD5");
            equalDigest = md.isEqual(MD5Digest.getBytes(),
contents.toString().getBytes());
            if (equalDigest == true){
                JOptionPane.showMessageDialog(null,"They are equal","MD5
Compare",JOptionPane.INFORMATION_MESSAGE);
            } else {
                JOptionPane.showMessageDialog(null,"They are not equal","SHA1
Compare",JOptionPane.WARNING_MESSAGE);
            }
        } catch (Exception e){
            System.out.println(e);
        }
    }

   /**This method compares the current SHA1 digest with a SHA1 digest
     * that is located in a file */
    private void menuItemCompSHA1ActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser chooser = new JFileChooser();
        File SHA1Dir = new File(pathname);
        File SHA1OutputFileTemp = new File(pathname, SHA1Dir.getName() + ".SHA");
        chooser.setCurrentDirectory(SHA1Dir);
        chooser.setSelectedFile(SHA1OutputFileTemp);
        int returnVal = chooser.showOpenDialog(null);

        if (returnVal == chooser.CANCEL_OPTION){
            return;
        }
        StringBuffer contents = new StringBuffer();
        BufferedReader input = null;
        try {
            input = new BufferedReader( new
FileReader(chooser.getSelectedFile().getAbsolutePath()));
            String line = null; //not declared within while loop

            while (( line = input.readLine()) != null){
                contents.append(line);
            }
        }catch (FileNotFoundException ex) {
            ex.printStackTrace();
        }catch (IOException ex){
              ex.printStackTrace();
        }
        finally {
            try {
                if (input!= null) {
                //flush and close both "input" and its underlying FileReader
                input.close();
                }
            }catch (IOException ex) {
                    ex.printStackTrace();
            }
        }

        MessageDigest md = null;
        boolean equalDigest;
        try{
            MessageDigest.getInstance("SHA1");
            equalDigest = md.isEqual(SHA1Digest.getBytes(),
contents.toString().getBytes());
            if (equalDigest == true){
                JOptionPane.showMessageDialog(null,"The digests match","SHA1
Compare",JOptionPane.INFORMATION_MESSAGE);
            } else {
                JOptionPane.showMessageDialog(null,"The digests do not match","SHA1
Compare",JOptionPane.WARNING_MESSAGE);
            }
        } catch (Exception e){
            System.out.println(e);
        }

    }

        /** This method saves the MD5 digest to a file */
    private void menuItemSaveMD5ActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser chooser = new JFileChooser();
        File MD5Dir = new File(pathname);
        File MD5OutputFileTemp = new File(pathname, MD5Dir.getName() + ".MD5");
        chooser.setCurrentDirectory(MD5Dir);
        chooser.setSelectedFile(MD5OutputFileTemp);
        int returnVal = chooser.showSaveDialog(null);
        File MD5OutputFile = new File(chooser.getSelectedFile().getAbsolutePath());

        //Check to see if file exists and if it does ask user if he/she indeed wants to
overwrite file
        if (MD5OutputFile.isFile()){
            if(JOptionPane.showConfirmDialog(null, "File Exists. Overwrite File?",
"Confim", JOptionPane.YES_NO_OPTION) != JOptionPane.YES_OPTION){
                this.menuItemSaveMD5ActionPerformed(evt);
                return;
            }
        }

        //If user cancels, then just return
        if (returnVal == chooser.CANCEL_OPTION){
            return;
        }

        //Write throws eoutput file
        try {
            BufferedWriter out = new BufferedWriter(new
FileWriter(chooser.getSelectedFile().getAbsolutePath()));
            out.write(MD5Digest);
            out.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }

    /** This method saves the SHA1 digest to a file */
    private void menuItemSaveSHA1ActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser chooser = new JFileChooser();
        File SHA1Dir = new File(pathname);
        File SHA1OutputFileTemp = new File(pathname, SHA1Dir.getName() + ".SHA");
        chooser.setCurrentDirectory(SHA1Dir);
        chooser.setSelectedFile(SHA1OutputFileTemp);
        int returnVal = chooser.showSaveDialog(null);
        File SHA1OutputFile = new File(chooser.getSelectedFile().getAbsolutePath());

        //Check to see if file exists and if it does ask user if he/she indeed wants to
overwrite file
        if (SHA1OutputFile.isFile()){
            if(JOptionPane.showConfirmDialog(null, "File Exists. Overwrite File?",
"Confim", JOptionPane.YES_NO_OPTION) != JOptionPane.YES_OPTION){
                this.menuItemSaveSHA1ActionPerformed(evt);
                return;
            }
        }

        if (returnVal == chooser.CANCEL_OPTION){
            return;
        }

        try {
            BufferedWriter out = new BufferedWriter(new
FileWriter(chooser.getSelectedFile().getAbsolutePath()));
            out.write(SHA1Digest);
            out.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }

    /** This method closes the application */
    private void menuItemExitActionPerformed(java.awt.event.ActionEvent evt) {
        System.exit(0);
    }

    /** This method changes the MD5 text box to hex or back */
    private void chkHexMD5ActionPerformed(java.awt.event.ActionEvent evt) {
        if(chkHexMD5.isSelected()){
            txtMD5.setText(CryptUtil.byteArray2Hex(txtMD5.getText()));
        }else {
            txtMD5.setText(MD5Digest);
        }
    }

    /** This method changes the SHA1 text box to hex or back */
    private void chkHexSHA1ActionPerformed(java.awt.event.ActionEvent evt) {
        if(chkHexSHA1.isSelected()){
            txtSHA1.setText(CryptUtil.byteArray2Hex(txtSHA1.getText()));
        }else {
            txtSHA1.setText(SHA1Digest);
        }
    }

    /** This method gets the MD5 of the file loaded */
    private void menuItemGetMD5ActionPerformed(java.awt.event.ActionEvent evt) {
        CryptDigest myDigest = new CryptDigest();
        MD5Digest = myDigest.computeDigest(pathname, "MD5");
        txtMD5.setText(MD5Digest);
    }

    /** This method gets the SHA1 of the file loaded */
    private void menuItemGetSHA1ActionPerformed(java.awt.event.ActionEvent evt) {
        CryptDigest myDigest = new CryptDigest();
        SHA1Digest = myDigest.computeDigest(pathname, "SHA1");
        txtSHA1.setText(SHA1Digest);
    }

    /** This method gets an input file from the user with a JFileChooser */
    private void menuItemOpenFileActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser chooser = new JFileChooser();
        int returnVal = chooser.showOpenDialog(null);

        if (returnVal == chooser.CANCEL_OPTION){   //exit method if user selects "Cancel"
            return;
        }
        pathname = chooser.getSelectedFile().getAbsolutePath();   //set pathname for file
to work with

        //reset digest values and text box values to null when we select a new file
        txtInFilePathName.setText(pathname);
        txtMD5.setText(null);
        txtSHA1.setText(null);
        MD5Digest = null;
        SHA1Digest = null;
    }


    /** main method  */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new CryptMain().setVisible(true);
            }
        });
    }


    // Variables declaration - do not modify
    private javax.swing.JCheckBox chkHexMD5;
    private javax.swing.JCheckBox chkHexSHA1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JMenu jMenu1;
    private javax.swing.JMenu jMenu2;
    private javax.swing.JMenu jMenu3;
    private javax.swing.JMenu jMenu4;
    private javax.swing.JMenu jMenu5;
    private javax.swing.JMenu jMenu6;
    private javax.swing.JMenu jMenu7;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JPanel jPanel4;
    private javax.swing.JTabbedPane jTabbedPane1;
    private javax.swing.JMenuItem menuItemAddKey;
    private javax.swing.JMenuItem menuItemCompMD5;
    private javax.swing.JMenuItem menuItemCompSHA1;
    private javax.swing.JMenuItem menuItemCreateKeyStore;
    private javax.swing.JMenuItem menuItemDecryptFile;
    private javax.swing.JMenuItem menuItemDeleteKey;
    private javax.swing.JMenuItem menuItemEncryptFile;
    private javax.swing.JMenuItem menuItemExit;
    private javax.swing.JMenuItem menuItemGetBoth;
    private javax.swing.JMenuItem menuItemGetMD5;
    private javax.swing.JMenuItem menuItemGetSHA1;
    private javax.swing.JMenuItem menuItemOpenFile;
    private javax.swing.JMenuItem menuItemSaveMD5;
    private javax.swing.JMenuItem menuItemSaveSHA1;
    private javax.swing.JMenuItem menuItemViewAliases;
    private javax.swing.JTextArea txtAliases;
    private javax.swing.JTextField txtInFilePathName;
    private javax.swing.JTextField txtKeyStorePath;
    private javax.swing.JTextField txtMD5;
    private javax.swing.JTextField txtSHA1;
    // End of variables declaration

}
```

****
* Source Code for CryptDigest
****
```
/*
 * objCrypt.java
 *
 * Created on January 21, 2006, 5:18 PM
 *
 * To change this template, choose Tools | Options and locate the template under
 * the Source Creation and Management node. Right-click the template and choose
 * Open. You can then make changes to the template in the Source Editor.
 */

package crypt;
import java.security.MessageDigest;
import java.io.*;

/**
 *
 * @author Nick
 */
public class CryptDigest {

    /** Creates a new instance of object computDigest */
    public CryptDigest() {
    }

    String computeDigest(String inFile, String algorithm){
        File datafile = new File(inFile);
        String digest = new String();

        try {
            MessageDigest md = MessageDigest.getInstance(algorithm);
            FileInputStream fis = new FileInputStream(datafile);
            byte[] dataBytes = new byte[1024];
            int nread = fis.read(dataBytes);
            while (nread > 0) {
                md.update(dataBytes, 0, nread);
                nread = fis.read(dataBytes);
             }
             byte[] mdbytes = md.digest();
             System.out.println("Digest: " + mdbytes);
             //digest = new String(Util.byteArray2Hex(mdbytes));
             digest = new String(mdbytes);

        }catch (Exception e){
            System.out.println(e);
        }

        return digest;
    }
}
```

****
* Source Code for CryptKeyStore
****
```
/*
 * crypto.java
 *
 * Created on April 20, 2006, 9:03 PM
 *
 * To change this template, choose Tools | Options and locate the template under
 * the Source Creation and Management node. Right-click the template and choose
 * Open. You can then make changes to the template in the Source Editor.
 */

package crypt;
```

```
import java.security.KeyStore;
import java.security.KeyStore.PasswordProtection;
import java.util.Enumeration;
import javax.crypto.SecretKey;
import javax.swing.JOptionPane;

/**
 *
 * @author Nick
 */
public class CryptKeyStore {

    /** Creates a new instance of kStore */
    public CryptKeyStore() {
    }

    /** This method is used to create a new kesytore.  */
    public int createKeystore(String keyFilePath, String password){
        KeyStore ks = null;
        int result;
        /*Create concrete KeyStore instance and then
         *Load needs to be done before accessing the keystore.  In
         *this case we are creating it empty by passing in null.
         *We then store the file and close it*/
        try {
            ks = KeyStore.getInstance("JCEKS");
            ks.load(null, password.toCharArray());
            java.io.FileOutputStream fos = new java.io.FileOutputStream(keyFilePath);
            ks.store(fos, password.toCharArray());
            fos.close();
            result = 1;
//success
        }catch (Exception e){
            System.out.println(e);
            result = 0;                                                     //failure
        }

        return result;
    }

    /** This method is used to add a key to a current keystore  */
    public int addKey(String keyFilePath, String password, SecretKey sKey, String alias){
        KeyStore ks = null;  //creates the keystore
        int result;
        PasswordProtection pass = new PasswordProtection(password.toCharArray());
//creates the password protection for the keystore
        /*Initialize (load) the keystore*/
        try {
            ks = this.initKeystore(keyFilePath, password);
            KeyStore.SecretKeyEntry skEntry = new KeyStore.SecretKeyEntry(sKey);
            ks.setEntry(alias, skEntry, pass);
            java.io.FileOutputStream fos = new java.io.FileOutputStream(keyFilePath);
            ks.store(fos, password.toCharArray());
            fos.close();

            result = 1;                                                     //success
        }catch (Exception e){
            result= 0;                                                      //failure
            System.out.println(e);
        }

        return result;
    }

    /** This method is used to add a key to a current keystore  */
    public int deleteKey(String keyFilePath, String password, String alias){
        KeyStore ks = null;  //creates the keystore
        int result;
        PasswordProtection pass = new PasswordProtection(password.toCharArray());
//creates the password protection for the keystore
        /*Initialize (load) the keystore*/
        try {
            ks = this.initKeystore(keyFilePath, password);
            ks.deleteEntry(alias);
            java.io.FileOutputStream fos = new java.io.FileOutputStream(keyFilePath);
            ks.store(fos, password.toCharArray());
            fos.close();

            result = 1;                                                         //success
        }catch (Exception e){
            result= 0;                                                          //failure
            System.out.println(e);
        }

        return result;
    }


    /** This method is used to retrieve a key from a current keystore  */
    public SecretKey getKey(String keyFilePath, String password, String alias){
        System.out.println("getKey");
        KeyStore ks = null;         //creates the keystore
        SecretKey sKey = null;      //creates the secret key that will be returned
        PasswordProtection pass = new PasswordProtection(password.toCharArray());
//creates the password protection for the keystore
        /*Initialize (load) the keystore*/
        try {
            ks = this.initKeystore(keyFilePath, password);
            KeyStore.SecretKeyEntry skEntry = (KeyStore.SecretKeyEntry)
ks.getEntry(alias, pass);
            sKey = skEntry.getSecretKey();
        }catch (Exception e){
            System.out.println(e);
        }

        System.out.println(sKey.getEncoded());
        return sKey;
    }

    /** This method is a private method that is used to initialize (load) the keystore
*/
    private KeyStore initKeystore(String keyFilePath, String password){
        System.out.println("initKeystore");
        KeyStore ks = null;
        /*Loads the keystore with the given pathname and password*/
        try {
            ks = KeyStore.getInstance("JCEKS");
            java.io.FileInputStream fis = new java.io.FileInputStream(keyFilePath);
            ks.load(fis, password.toCharArray());
            fis.close();

        }catch (Exception e){
            System.out.println(e);
        }


        return ks;
    }

    public Enumeration getAliases(String keyFilePath, String password){
        KeyStore ks = null;  //creates the keystore
        Enumeration aliases = null;
        PasswordProtection pass = new PasswordProtection(password.toCharArray());
//creates the password protection for the keystore
        /*Initialize (load) the keystore*/
        try {
            ks = this.initKeystore(keyFilePath, password);
            aliases = ks.aliases();
        }catch (Exception e){
            System.out.println(e);
            JOptionPane.showMessageDialog(null,"Failed to retrieve Aliases","Key
Store",JOptionPane.WARNING_MESSAGE);
```

```
        }

        return aliases;
    }

}
```

****
* Source Code for CryptCipher
****

```
/*
 * cryptCipher.java
 *
 * Created on April 21, 2006, 9:28 PM
 *
 * To change this template, choose Tools | Options and locate the template under
 * the Source Creation and Management node. Right-click the template and choose
 * Open. You can then make changes to the template in the Source Editor.
 */

package crypt;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import javax.crypto.Cipher;
import javax.crypto.CipherInputStream;
import javax.crypto.CipherOutputStream;
import javax.crypto.SecretKey;

/**
 *
 * @author Nick
 */
public class CryptCipher {

    /** Creates a new instance of cryptCipher */
    public CryptCipher() {
    }

    /** This method is used to encrypt a file using DES */
    public int encrypt(String inFilePath, String outFilePath, SecretKey sKey){
        System.out.println("encrypt");
        int result;
        FileInputStream inputFile;
        FileOutputStream outputFile;
        CipherOutputStream cipherOutputStream;
        Cipher myCipher;

        try {
            myCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
//creating the concrete cipher
            myCipher.init(Cipher.ENCRYPT_MODE, sKey);
//initializing the cipher for encryption
            inputFile = new FileInputStream(inFilePath);
//creating input file stream
            outputFile = new FileOutputStream(outFilePath);
//creating output file stream
            cipherOutputStream = new CipherOutputStream(outputFile, myCipher);  //create
output cipher stream

            int i = 0;
            while ((i = inputFile.read()) != -1) {
                cipherOutputStream.write(i);
            }

            cipherOutputStream.close();
            outputFile.close();
            inputFile.close();
            result = 1;                                                       //success
        }catch (Exception e){
            result = 0;                                                 //failure
            System.out.println(e);
        }

        return result;

    }

    /** This method is used to decrypt a file using DES */
    public int decrypt(String inFilePath, String outFilePath, SecretKey sKey){
        System.out.println("decrypt");
        int result;
        FileInputStream inputFile;
        FileOutputStream outputFile;
        CipherInputStream cipherInputStream;
        Cipher myCipher;

        try {
            myCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
//creating the concrete cipher
            myCipher.init(Cipher.DECRYPT_MODE, sKey);
//initializing the cipher for decryption
            inputFile = new FileInputStream(inFilePath);
//creating input file stream
            outputFile = new FileOutputStream(outFilePath);
//creating output file stream
            cipherInputStream = new CipherInputStream(inputFile, myCipher);     //create
input cipher stream

            int i = 0;
            while ((i = cipherInputStream.read()) != -1) {
                outputFile.write(i);
            }

            cipherInputStream.close();
            outputFile.close();
            inputFile.close();
            result = 1;                                                 //success
        }catch (Exception e){
            result = 0;                                                 //failure
            System.out.println(e);
        }

        return result;

    }
}
```