

## Java Cryptography

Dr. Guillermo A. Francia, III

## Java Cryptography Extension (JCE)

- JCE is a set of packages that provides a framework and implementations for:
  - Encryption. Support for encryption includes symmetric, asymmetric, block, and stream ciphers.
  - Key generation and key agreement
  - Message Authentication Code (MAC) algorithms
- The software also supports secure streams and sealed objects.

## Java Cryptography Provider

- Download and install a Java cryptography provider API
  - copy `cryptix-jce-provider.jar` in `%Java_JDK_Install_Folder%\jre\lib\ext`
- Import the following

```
javax.crypto.spec.IvParameterSpec;
java.security.Key;
javax.crypto.CipherInputStream;
javax.crypto.CipherOutputStream;
java.security.Provider;
java.security.SecureRandom;
java.security.Security;
cryptix.jce.provider.CryptixCrypto;
javax.crypto.Cipher;
javax.crypto.KeyGenerator;
javax.crypto.SecretKey;
java.security.MessageDigest;
sun.misc.BASE64Encoder;
```

## Install the Provider

- Perform a dynamic installation

```
private void installProvider() {
    Provider cryptix_provider = new CryptixCrypto();
    int result =
        Security.addProvider(cryptix_provider);
    if (result < 0) {
        JOptionPane.showMessageDialog(this,
            "Cryptographic provider failed", "ERROR",
            JOptionPane.ERROR_MESSAGE);
    }
    else installedProvider = true;
}
```

## Key Generator and Key Length

- Set the key length and generate the key

```
Cipher cipher = Cipher.getInstance(alg+"/"+ mode +
"/" + padding ,provider);

KeyGenerator kg =
KeyGenerator.getInstance(alg,provider);
if (alg.equals("DES"))          keyLength=56;
else if (alg.equals("RC2"))     keyLength=128;
else if (alg.equals("DESede"))  keyLength=56*3;
else if (alg.equals("Rijndael")) keyLength=128;
else return;

kg.init(keyLength, new SecureRandom());

SecretKey key = kg.generateKey();
```

## Initialization Vector (IV)

- Set the Initialization Vector (IV)

```
cipher.init(Cipher.ENCRYPT_MODE, key);
newIV = new byte[cipher.getBlockSize()];
char [] password = "password"; //or ask the user

//obfuscate the IV with the password by XOR
for (int i=0; i<password.length; i++)
    newIV[i] = (byte) ((int) newIV[i] ^
(int) password[i]);
```

## Append the IV

```
// Create new cipher object using
// algorithm/mode/padding stored
cipher=Cipher.getInstance(alg, provider);
// Check if IV is needed.
if (mode.equals("ECB"))
    cipher.init(method, secretKey);
else {
    ivSpec = new IvParameterSpec(iv);
    cipher.init(method, secretKey, ivSpec);
}
```

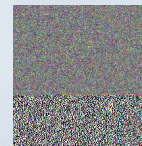
## Comparing Encryption Methods



Original



Encrypted using ECB mode



Encrypted using other mode

Source: [http://en.wikipedia.org/wiki/Block\\_cipher\\_modes\\_of\\_operation](http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation)

## Encryption

```
// Read file to encrypt
fInput = new FileInputStream(inputFile);
// Set output file.
fOutput = new FileOutputStream(outputFile+".encrypted");
// Create CipherOutputStream using cipher "cipher".
cStr = new CipherOutputStream(fOutput, cipher);
// Read input bytes into buffer and run them through the cipher stream.
byte[] buffer = new byte[8192];
int length=0;      long totRead =0;
while ((length=fInput.read(buffer))!= -1){
    totRead += length;
    cStr.write(buffer, 0, length);
}
// Close Streams.
fInput.close();
cStr.close();
```

## Saving the Key+IV in a File

```
private boolean saveKeyWithIV() {
    FileOutputStream fOut = null;  boolean result=false;
    try {
        fOut = new FileOutputStream(keyfile);
        ObjectOutputStream objStr=new ObjectOutputStream(fOut);
        objStr.writeObject(secretKey);
        if (!mode.equals("ECB")) // Write the IV after the key.
            fOut.write(iv);
        objStr.close();      fOut.close();
        result = true;
    }
    catch (Exception ex){ System.out.println("Exception!\n");
    }
    return result;
} //end of saveKeyWithIV
```

## Retrieving the Key+IV from a File

```
private boolean getKeyWithIV() {
    boolean result=false;      FileInputStream fInput = null;
    try {
        // Create FileInputStream on keyfile.
        fInput = new FileInputStream(keyfile);
        ObjectInputStream objIStream = new ObjectInputStream(fInput);
        // Read key object from file.
        secretKey=(Key) objIStream.readObject();
        // Read the IV from the file.
        iv = new byte[fInput.available()];
        if (!mode.equals("ECB"))
            fInput.read(iv);
        objIStream.close();      result=true;
    }
    catch (Exception ex){ ex.printStackTrace(); }
    return result;
} //end of getKeyWithIV
```

## Decryption

```
CipherInputStream ciStr=null;
// Open file containing encrypted data.
fInput = new FileInputStream(outputFile+".encrypted");
ciStr = new CipherInputStream(fInput, cipher);
fOutput = new FileOutputStream(outputFile+".decrypted");
// Read bytes and run them through cipher and store them to
// file back again.
byte[] buffer=new byte[8192];
int length=0;      long totRead =0;
while ((length=ciStr.read(buffer))!=-1){
    totRead += length;
    fOutput.write (buffer, 0, length);
}
// Close streams.
ciStr.close();
fOutput.close();
```

## Message Digest

```
MessageDigest md=null;
byte[] hash_raw=null;
try {
    md = MessageDigest.getInstance(hashDigest, provider); }
catch (Exception ex) { ex.printStackTrace();}
FileInputStream fis = null;
try {
    fis = new FileInputStream(inputFile);
    byte[] buffer = new byte [8192];
    int length=0;
    while ((length = fis.read(buffer)) != -1)
        md.update(buffer, 0, length);

    hash_raw = md.digest();
}
catch (Exception ex) { ex.printStackTrace(); }

BASE64Encoder enc = new BASE64Encoder();
String base64 = enc.encode(hash_raw);
```