My Crypto Libraries
HW4 - CNS Sapienza

Edoardo Puglisi 1649359

28/11/2019

Contents

1 Overview

Following sections will describe two different crypto libraries for two different programming languages, Java and Python.

2 Java

The first programming language we are going to describe is Java. The two libraries used are *Javax.crypto* and *Apache.commons.crypto*.

2.1 Javax.Crypto

This is the default crypto-library for the Java Platform and provides the classes and interfaces for cryptographic operations used by other libraries. The license type change according to the SDK type: Oracle OpenJDK is under GPLv2 License while Oracle JDK requires a commercial license in production.

Encryption This is an example of encryption for a given array of bytes. First we create an initalization vector *iv_spec* to be used later for the cipher's algorithm.

```
IvParameterSpec iv_spec = new IvParameterSpec (iv);
```

IvParameterSpec constructor takes as input an array of bytes(*iv*) and returns an object of class *IvParameterSpec* that specify the Initalization Vector.
Now we generate the secret key *secret_key_spec*.

```
SecretKeySpec secret_key_spec = new SecretKeySpec(
    secret_key , "AES");
```

SecretKeySpec constructor takes in input the user's secret key (*secret_key*) and the algorithm to be associated with it and returns an object of class *SecretKeySpec* that specify the secret key used by the cipher.
With the key and the iv we can initialize the cipher for the encryption.

```
Cipher cipher = Cipher.getInstance("AES/CBC/
    PKCS5PADDING");
cipher.init(Cipher.ENCRYPT_MODE, secret_key_spec ,
    iv_spec);
return cipher.doFinal(to_encrypt);
```

Cipher.getInstance generates a cipher for the given type "algorithm/mode/padding" in this case AES is the algorithm, CBC the operating mode and PKCS5PADDING the type of padding used. Then we initialize the cipher with the key and iv we previously created and set the cipher mode for the encryption. When the cipher is initialized correctly we can start the ecnryption with the *.doFinal*() method that takes as input the plaintext's bytes and returns a sequence of bytes.

Decryption The operations for the decryption are pretty similar to the encryption's ones. The only difference is in the cipher configuration where we change ENCRYPT_MODE with DECRYPT_MODE

```
cipher.init(Cipher.DECRYPT_MODE, secret_key_spec,
    iv_spec);
```

2.2 Apache.Commons.crypto

The Apache Commons is a project of the Apache Software Foundation with he purpose of provide reusable, open source Java software.

Encryption The procedure starts exactly as before, with the creation of IV and secretkey.

```
IvParameterSpec iv_spec = new IvParameterSpec(iv);
SecretKeySpec secret_key_spec = new SecretKeySpec(
    secret_key, "AES");
```

Then we initialize the properties for the Cipher.

```
Properties properties = new Properties();
properties.setProperty(CryptoCipherFactory.CLASSES_KEY
    , CipherProvider.OPENSSL.getClassName());
CryptoCipher cipher = Utils.getCipherInstance("AES/CBC
    /PKCS5Padding", properties);
cipher.init(Cipher.ENCRYPT_MODE, secret_key_spec,
    iv_spec);
```

Class CryptoCipher takes in input the same "algorithm/mode/padding" string as Javax.crypto and the properties we defined for it.
Method *.setProperty* initializes a pair key-value: *CLASS_KEY* value must be a class that implements the CryptoCipher interface: *OPENSSL*. At last we execute the encryption procedure.

```
cipher.doFinal(to_encrypt, 0, to_encrypt.length, out,
    0);
return out;
```

Method *.doFinal*() for *CryptoCipher* class takes as parameters the input (in bytes), the starting index and its length plus the output array in which will be copied returning the number of bytes generated.

Decryption Also in this library the decryption procedure change only on the cipher's configuration phase.

```
cipher.init(Cipher.DECRYPT_MODE, secret_key_spec,
    iv_spec);
```

2.3 Comparison

Using the following code we demonstrated that the two libraries are interchangeable without any problem. According to usage simplicity the integrated *javax.crypto* library is recommended.

```
import java.util.Arrays;
import java.util.Base64;
import java.util.Properties;
import java.util.Random;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import org.apache.commons.crypto.cipher.CryptoCipher;
import org.apache.commons.crypto.cipher.
    CryptoCipherFactory;
import org.apache.commons.crypto.cipher.
    CryptoCipherFactory.CipherProvider;
import org.apache.commons.crypto.utils.Utils;

public class script_java {

    public static byte[] encrypt(byte[] iv, byte[] pw,
         byte[] msg) {
         try {
              IvParameterSpec iv_spec = new
                  IvParameterSpec(iv);
              SecretKeySpec pw_spec = new SecretKeySpec(
                  pw, "AES");
              Cipher cipher = Cipher.getInstance("AES/
                  CBC/PKCS5PADDING");
              cipher.init(Cipher.ENCRYPT_MODE, pw_spec,
                  iv_spec);
              return cipher.doFinal(msg);
         } catch (Exception ex) {
              ex.printStackTrace();
         }
         return null;
```

```
}


public static byte[] decrypt(byte[] iv, byte[] pw,
    byte[] enc_msg) {
    try {
        IvParameterSpec iv_spec = new
           IvParameterSpec(iv);
        SecretKeySpec pw_spec = new SecretKeySpec(
           pw, "AES");
        Cipher cipher = Cipher.getInstance("AES/
           CBC/PKCS5PADDING");
        cipher.init(Cipher.DECRYPT_MODE, pw_spec,
           iv_spec);
        return cipher.doFinal(enc_msg);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    return null;
}

public static byte[] encrypt_apache(byte[] iv,
   byte[] pw, byte[] msg) {
    try {
        IvParameterSpec iv_spec = new
           IvParameterSpec(iv);
        SecretKeySpec pw_spec = new SecretKeySpec(
           pw, "AES");
        Properties properties = new Properties();
        properties.setProperty(CryptoCipherFactory
           .CLASSES_KEY, CipherProvider.OPENSSL.
           getClassName());
        CryptoCipher cipher = Utils.
           getCipherInstance("AES/CBC/PKCS5Padding
           ", properties);
        cipher.init(Cipher.ENCRYPT_MODE, pw_spec,
           iv_spec);
        byte[] out = new byte[msg.length * 2];
        int bytes = cipher.doFinal(msg, 0, msg.
           length, out, 0);
        return Arrays.copyOf(out, bytes);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    return null;
}

public static byte[] decrypt_apache(byte[] iv,
   byte[] pw, byte[] enc_msg) {
    try {
        IvParameterSpec iv_spec = new
           IvParameterSpec(iv);
        SecretKeySpec pw_spec = new SecretKeySpec(
           pw, "AES");
        Properties properties = new Properties();
        properties.setProperty(CryptoCipherFactory
           .CLASSES_KEY, CipherProvider.OPENSSL.
           getClassName());
        CryptoCipher cipher = Utils.
           getCipherInstance("AES/CBC/PKCS5Padding
           ", properties);
        cipher.init(Cipher.DECRYPT_MODE, pw_spec,
           iv_spec);
        byte[] out = new byte[enc_msg.length * 2];
        int bytes = cipher.doFinal(enc_msg, 0,
           enc_msg.length, out, 0);
        return Arrays.copyOf(out, bytes);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    return null;
}

public static void main(String[] args) {

    Random rd = new Random();
    byte[] iv = new byte[16];
    rd.nextBytes(iv);
    byte[] pw = new byte[32];
    rd.nextBytes(pw);
    int length = (int) Math.pow((double) (10), (
       double) (8));
    byte[] data = new byte[length];
    rd.nextBytes(data);

    System.out.println("Plaintext:␣" + Base64.
       getEncoder().encodeToString(data).substring
       (0, 100) + "...");
```

```
        byte[] ciphertext = encrypt(iv, pw, data);
        System.out.println("Ciphertext␣with␣javax.
            crypto:␣" + Base64.getEncoder().
            encodeToString(ciphertext).substring(0,
            100) + "...");
        byte[] plaintext = decrypt_apache(iv, pw,
            ciphertext);
        System.out.println("Decrypted␣with␣org.apache.
            commons.crypto:␣" + Base64.getEncoder().
            encodeToString(plaintext).substring(0, 100)
             + "...");
        System.out.println("Compare␣1␣=␣"+ Arrays.
            equals(data, plaintext) );
        data = new byte[length];
        byte[] ciphertext = encrypt_apache(iv, pw,
            data);
        System.out.println("Ciphertext␣with␣org.apache
            .commons.crypto:␣" + Base64.getEncoder().
            encodeToString(ciphertext).substring(0,
            100) + "...");
        byte[] plaintext = decrypt(iv, pw, ciphertext)
            ;
        System.out.println("Decrypted␣with␣javax.
            crypto::␣" + Base64.getEncoder().
            encodeToString(plaintext).substring(0, 100)
             + "...");
        System.out.println("Compare␣2␣=␣"+ Arrays.
            equals(data, plaintext) );



    }
}
```

3 Python

The two libraries used in Python are *pycrypto* and *pyca/cryptography* both open source projects. In python, API are much easier than java, reducing the programmer job to few lines of code.

3.1 Pycrypto

First of all we must import the cipher for the algorithm we want to use from the library, in our case AES.

```
from Crypto.Cipher import AES
```

For both encryption and decryption the procedure is the same: create the cipher with function *new* giving as input the key, the operating mode, and the initialization vector. Key can be 16/24/32 bits long while the IV must be 16 bits long. Then execute the encrypt/decrypt function.

Encryption

```
aes = AES.new(key, AES.MODE_CBC, iv)
ciphertext = aes.encrypt(data)
```

Decryption

```
aes = AES.new(key, AES.MODE_CBC, iv)
plaintext = aes.decrypt(ciphertext)
```

3.2 Pyca/cryptography

The *cryptography* library was originally designed to support multiple backends and for this reason this kind of objects provide multiple ciphers used for encryption and decryption. Lets start by importing the default backend and the modules we will need for cryptography such as algorithms, modes and ciphers primitives.

```
from cryptography.hazmat.backends import
    default_backend
from cryptography.hazmat.primitives.ciphers import
    Cipher, algorithms, modes
```

Initialize cipher.

```
backend = default_backend()
cipher = Cipher(algorithms.AES(key), modes.CBC(iv),
    backend=backend)
```

Key and IV length constraints are the same as previous library. The backend has the purpose to provide methods for using ciphers for encryption and decryption.

Encryption

```
encryptor = cipher.encryptor()
ciphertext = encryptor.update(data) + encryptor.
    finalize()
```

Decryption

```
decryptor = cipher.decryptor()
plaintext = decryptor.update(ciphertext) + decryptor.
    finalize()
```

Encryption and decryption phases differ only for the use of encryptor for the first and decryptor for the second. When calling encryptor() or decryptor() on a Cipher object the result will conform to the CipherContext interface. We then call update(data) with data until we have fed everything into the context. Once that is done we call finalize() to finish the operation and obtain the remainder of the data.

3.3 Comparison

In python too both libraries are interchangeable. The pyca/cryptography though is the most up-to-date between the two.

```
import base64
import os

from cryptography.hazmat.backends import
    default_backend
from cryptography.hazmat.primitives.ciphers import
    Cipher, algorithms, modes

from Crypto.Cipher import AES

pw = os.urandom(32)
msg = "Very useless message"
iv = os.urandom(16)

backend = default_backend()
cipher = Cipher(algorithms.AES(pw), modes.CBC(iv),
    backend=backend)
encryptor = cipher.encryptor()
```

```
ciphertext = encryptor.update(msg) + encryptor.
    finalize()

aes = AES.new(pw, AES.MODE_CBC, iv)
plaintext2 = aes.decrypt(ciphertext)

aes = AES.new(pw, AES.MODE_CBC, iv)
ciphertext2 = aes.encrypt(msg)


backend = default_backend()
cipher = Cipher(algorithms.AES(pw), modes.CBC(iv),
    backend=backend)
decryptor = cipher.decryptor()
plaintext = decryptor.update(ciphertext2) + decryptor.
    finalize()

print(f"Plaintext:␣{str(base64.b64encode(msg),␣'utf8')
    [:100]}...")
print(f"Ciphertext␣with␣cryptography:␣{str(base64.
    b64encode(ciphertext),␣'utf8')[:100]}...")
print(f"Decrypted␣decrypted␣with␣pycrypto:␣{str(base64
    .b64encode(plaintext2),␣'utf8')[:100]}...")
print(f"Compare␣1=␣?␣{plaintext2==msg}")

print(f"Ciphertext␣with␣pycrypto:␣{str(base64.
    b64encode(ciphertext2),␣'utf8')[:100]}...")
print(f"Decrypted␣decrypted␣with␣cryptography:␣{str(
    base64.b64encode(plaintext),␣'utf8')[:100]}...")
print(f"Compare␣2=␣?␣{plaintext==msg}")
```