Computer Cyber Security

Project 1

Cryptographic Hash Functions and Ciphers

10/10/2018

By: **Pranati Trivedi**

Acknowledgment: I acknowledge that all of the work including figures and codes belong to me and/or persons who are referenced.

Signature: PRANATI TRIVEDI

1. **Your source code. If third party libraries are used, please write instructions on how to install them.**

```
package iit.ece443.prj01;

import java.security.MessageDigest;
import java.util.Arrays;
import java.io.UnsupportedEncodingException;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.ShortBufferException;
import java.security.NoSuchAlgorithmException;
import javax.crypto.AEADBadTagException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.ShortBufferException;


import javax.crypto.Cipher;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;


public class Main
{
    public static void main(String[] args)
        throws Exception
    {
   AES256();
   System.out.println();
   AES512();
   System.out.println();
   verifyMD5();
   System.out.println();

   perfMD5();
   System.out.println();
   perf256();
   System.out.println();
   perf512();
   System.out.println();

   verifyAESGCM();
   System.out.println();
   System.out.println("\n Modified for Attack: ");
   String msg = "Hello world!";
   attackAESGCM(msg);
   System.out.println();
   verifyAESCBC(); // bonus question
   System.out.println();
    perfAESGCM();
   System.out.println();
   perfAESCBC();  // bonus question
```

```
}

private static void AES256()
    throws Exception
{
    MessageDigest md = MessageDigest.getInstance("sha-256");

    String str = "Hello world!";

    md.update(str.getBytes("UTF-8"));
    byte[] hash = md.digest();

    System.out.printf("AES256 of [%s]%n", str);
    System.out.printf("Computed: %s%n", hexString(hash));
}
private static void AES512()
    throws Exception
{
    MessageDigest md = MessageDigest.getInstance("sha-512");

    String str = "Hello world!";

    md.update(str.getBytes("UTF-8"));
    byte[] hash = md.digest();

    System.out.printf("AES512 of [%s]%n", str);
    System.out.printf("Computed: %s%n", hexString(hash));
}

private static String hexString(byte[] buf)
{
    StringBuilder sb = new StringBuilder();
    for (byte b: buf)
        sb.append(String.format("%02X", b));
    return sb.toString();
}

private static void verifyMD5()
    throws Exception
{
    MessageDigest md = MessageDigest.getInstance("MD5");

    String str = "Hello world!";
    String md5 = "86FB269D190D2C85F6E0468CECA42A20";

    md.update(str.getBytes("UTF-8"));
    byte[] hash = md.digest();

    System.out.printf("MD5 of [%s]%n", str);
    System.out.printf("Computed: %s%n", hexString(hash));
    System.out.printf("Expected: %s%n", md5);
}

private static void perfMD5()
    throws Exception
{
    int MB = 256;

    byte[] buf = new byte[MB*1024*1024];
    Arrays.fill(buf, (byte)0);
```

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

        long start = System.currentTimeMillis();
        md.update(buf);
        byte[] hash = md.digest();
        long stop = System.currentTimeMillis();

        System.out.printf("MD5 of %dMB 0x00%n", MB);
        System.out.printf("Computed: %s%n", hexString(hash));
        System.out.printf("Time used: %d ms%n", stop-start);
        System.out.printf("Performance: %.2f MB/s%n", MB*1000.0/(stop-start));
    }

    private static void perf256()
        throws Exception
    {
        int MB = 256;

        byte[] buf = new byte[MB*1024*1024];
        Arrays.fill(buf, (byte)0);

        MessageDigest md = MessageDigest.getInstance("SHA-256");

        long start = System.currentTimeMillis();
        md.update(buf);
        byte[] hash = md.digest();
        long stop = System.currentTimeMillis();

        System.out.printf("SHA-256 of %dMB 0x00%n", MB);
        System.out.printf("Computed: %s%n", hexString(hash));

        System.out.printf("Time used: %d ms%n", stop-start);
        System.out.printf("Performance: %.2f MB/s%n", MB*1000.0/(stop-start));
    }
    private static void perf512()
        throws Exception
    {
        int MB = 256;

        byte[] buf = new byte[MB*1024*1024];
        Arrays.fill(buf, (byte)0);

        MessageDigest md = MessageDigest.getInstance("SHA-512");

        long start = System.currentTimeMillis();
        md.update(buf);
        byte[] hash = md.digest();
        long stop = System.currentTimeMillis();

        System.out.printf("SHA-512 of %dMB 0x00%n", MB);
        System.out.printf("Computed: %s%n", hexString(hash));

        System.out.printf("Time used: %d ms%n", stop-start);
        System.out.printf("Performance: %.2f MB/s%n", MB*1000.0/(stop-start));
    }


    private static void verifyAESGCM()
        throws Exception
    {
        String msg = "Hello world!";
```

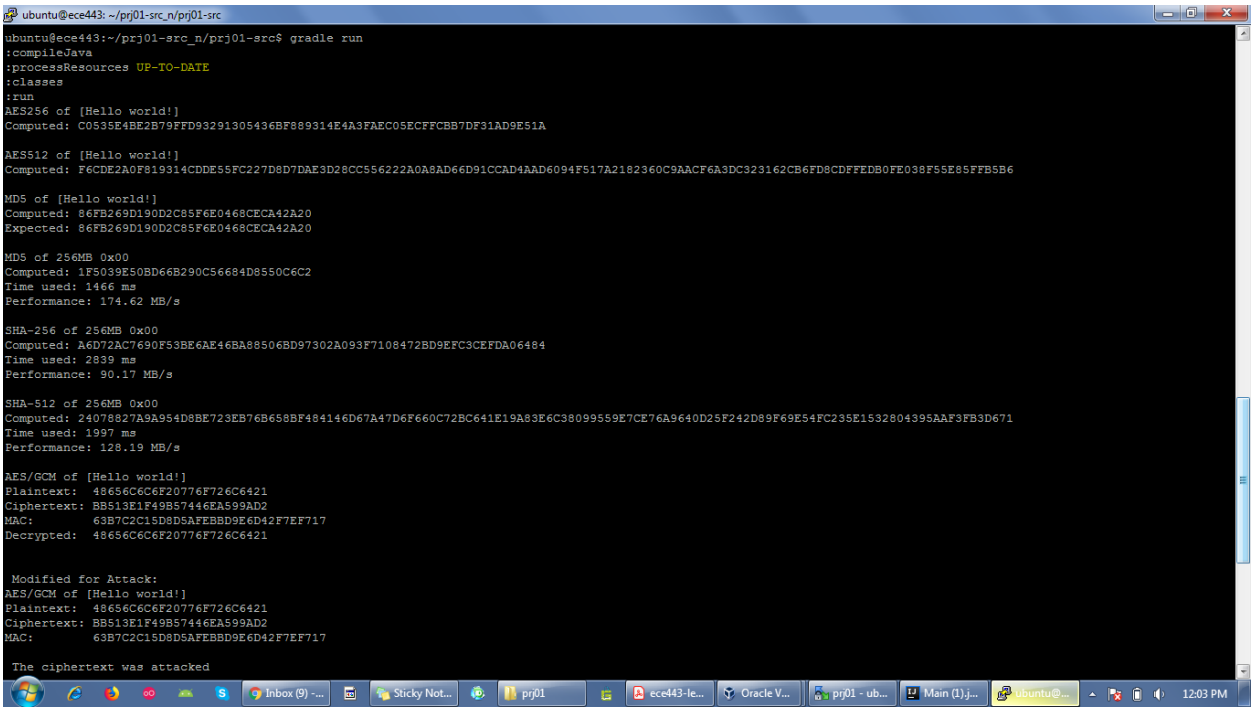
```
        byte[] buf = new byte[1000];

        byte[] iv = new byte[12];
        Arrays.fill(iv, (byte)0);
        GCMParameterSpec ivSpec = new GCMParameterSpec(128, iv);

        byte[] key = new byte[16];
        Arrays.fill(key, (byte)1);
        SecretKeySpec keySpec = new SecretKeySpec(key, "AES");

        Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");

        byte[] plaintext = msg.getBytes("UTF-8");

        cipher.init(Cipher.ENCRYPT_MODE, keySpec, ivSpec);
        int len = cipher.update(plaintext, 0, plaintext.length, buf);
        len += cipher.doFinal(buf, len);

        byte[] ciphertext = Arrays.copyOf(buf, len-16);
        byte[] mac = Arrays.copyOfRange(buf, len-16, len);

        System.out.printf("AES/GCM of [%s]%n", msg);
        System.out.printf("Plaintext:  %s%n", hexString(plaintext));
        System.out.printf("Ciphertext: %s%n", hexString(ciphertext));
        System.out.printf("MAC:        %s%n", hexString(mac));

        cipher.init(Cipher.DECRYPT_MODE, keySpec, ivSpec);
        int len2 = cipher.update(ciphertext, 0, ciphertext.length, buf);
        len2 += cipher.update(mac, 0, mac.length, buf, len2);
        len2 += cipher.doFinal(buf, len2);

        byte[] plaintext2 = Arrays.copyOf(buf, len2);
        System.out.printf("Decrypted:  %s%n", hexString(plaintext2));
    }

private static void attackAESGCM(String msg) throws NoSuchAlgorithmException,
NoSuchPaddingException, UnsupportedEncodingException, InvalidKeyException,
InvalidAlgorithmParameterException, ShortBufferException, IllegalBlockSizeException,
BadPaddingException
 {
    byte[] buf = new byte[1000];

     byte[] iv = new byte[12];
     Arrays.fill(iv, (byte)0);
     GCMParameterSpec ivSpec = new GCMParameterSpec(128, iv);

     byte[] key = new byte[16];
     Arrays.fill(key, (byte)1);
     SecretKeySpec keySpec = new SecretKeySpec(key, "AES");

     Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");

     byte[] plaintext = msg.getBytes("UTF-8");

     cipher.init(Cipher.ENCRYPT_MODE, keySpec, ivSpec);
     int len = cipher.update(plaintext, 0, plaintext.length, buf);
     len += cipher.doFinal(buf, len);

     byte[] ciphertext = Arrays.copyOf(buf, len-16);
     byte[] mac = Arrays.copyOfRange(buf, len-16, len);
```

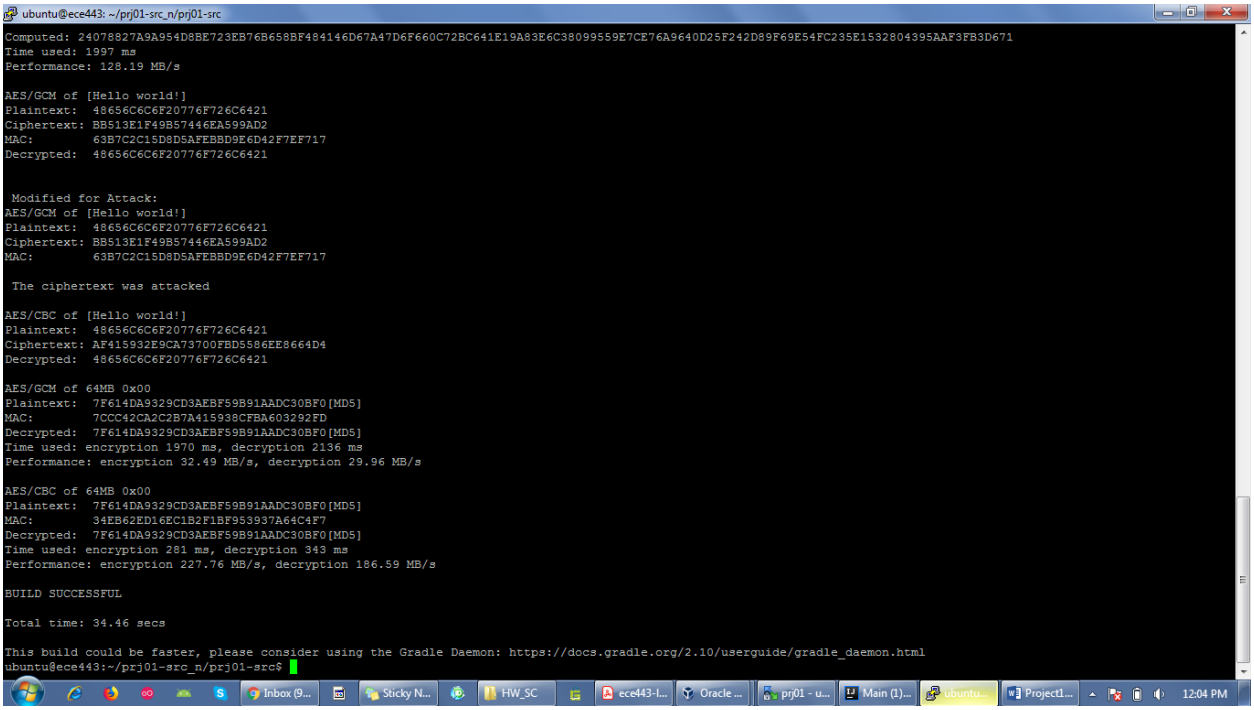
```
 System.out.printf("AES/GCM of [%s]%n", msg);
 System.out.printf("Plaintext:  %s%n", hexString(plaintext));
 System.out.printf("Ciphertext: %s%n", hexString(ciphertext));
 System.out.printf("MAC:        %s%n", hexString(mac));

 try{

     //Here I add new cipher to do attack
   String msgNew = "Cyber Attack happens";
  ciphertext = msgNew.getBytes("UTF-8");
  cipher.init(Cipher.DECRYPT_MODE, keySpec, ivSpec);
  int len2 = cipher.update(ciphertext, 0, ciphertext.length, buf);
       len2 += cipher.update(mac, 0, mac.length, buf, len2);
       len2 += cipher.doFinal(buf, len2);
       byte[] plaintext2 = Arrays.copyOf(buf, len2);
       System.out.printf("Decrypted:  %s%n", hexString(plaintext2));

 }catch(AEADBadTagException e) {

    System.out.println("\n The ciphertext was attacked ");

 }

}


private static void verifyAESCBC()
    throws Exception
{
    String msg = "Hello world!";
    byte[] buf = new byte[1000];

    byte[] iv = new byte[16];
    Arrays.fill(iv, (byte)0);
    IvParameterSpec IvParameterSpec = new IvParameterSpec(iv);

    byte[] key = new byte[16];
    Arrays.fill(key, (byte)1);
    SecretKeySpec keySpec = new SecretKeySpec(key, "AES");

    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

    byte[] plaintext = msg.getBytes("UTF-8");

    cipher.init(Cipher.ENCRYPT_MODE, keySpec, IvParameterSpec);
    int len = cipher.update(plaintext, 0, plaintext.length, buf);
    len += cipher.doFinal(buf, len);

    byte[] ciphertext = Arrays.copyOf(buf, len);

    System.out.printf("AES/CBC of [%s]%n", msg);
    System.out.printf("Plaintext:  %s%n", hexString(plaintext));
    System.out.printf("Ciphertext: %s%n", hexString(ciphertext));

    cipher.init(Cipher.DECRYPT_MODE, keySpec, IvParameterSpec);
    int len2 = cipher.update(ciphertext, 0, ciphertext.length, buf);
    len2 += cipher.doFinal(buf, len2);

    byte[] plaintext2 = Arrays.copyOf(buf, len2);
    System.out.printf("Decrypted:  %s%n", hexString(plaintext2));
}
```

```
    private static void perfAESGCM()
        throws Exception
    {
        int MB = 64;

        byte[] plaintext = new byte[MB*1024*1024];
        Arrays.fill(plaintext, (byte)0);

        byte[] buf = new byte[MB*1024*1024+16];

        byte[] iv = new byte[12];
        Arrays.fill(iv, (byte)0);
        GCMParameterSpec ivSpec = new GCMParameterSpec(128, iv);

        byte[] key = new byte[16];
        Arrays.fill(key, (byte)1);
        SecretKeySpec keySpec = new SecretKeySpec(key, "AES");

        Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");

        long startE = System.currentTimeMillis();
        cipher.init(Cipher.ENCRYPT_MODE, keySpec, ivSpec);
        int len = cipher.update(plaintext, 0, plaintext.length, buf);
        len += cipher.doFinal(buf, len);
        long stopE = System.currentTimeMillis();

        byte[] ciphertext = Arrays.copyOf(buf, len-16);
        byte[] mac = Arrays.copyOfRange(buf, len-16, len);

        System.out.printf("AES/GCM of %dMB 0x00%n", MB);
        System.out.printf("Plaintext:  %s[MD5]%n",
            hexString(MessageDigest.getInstance("MD5").digest(plaintext)));
        System.out.printf("MAC:        %s%n", hexString(mac));

        long startD = System.currentTimeMillis();
        cipher.init(Cipher.DECRYPT_MODE, keySpec, ivSpec);
        int len2 = cipher.update(ciphertext, 0, ciphertext.length, buf);
        len2 += cipher.update(mac, 0, mac.length, buf, len2);
        len2 += cipher.doFinal(buf, len2);
        long stopD = System.currentTimeMillis();

        byte[] plaintext2 = Arrays.copyOf(buf, len2);
        System.out.printf("Decrypted:  %s[MD5]%n",
            hexString(MessageDigest.getInstance("MD5").digest(plaintext2)));

        System.out.printf(
            "Time used: encryption %d ms, decryption %d ms%n",
            stopE-startE, stopD-startD);
        System.out.printf(
            "Performance: encryption %.2f MB/s, decryption %.2f MB/s%n",
            MB*1000.0/(stopE-startE), MB*1000.0/(stopD-startD));
    }
private static void perfAESCBC()
        throws Exception
    {
        int MB = 64;

        byte[] plaintext = new byte[MB*1024*1024];
```

```
        Arrays.fill(plaintext, (byte)0);

        byte[] buf = new byte[MB*1024*1024+16];

        byte[] iv = new byte[16];
        Arrays.fill(iv, (byte)0);
        IvParameterSpec ivSpec = new IvParameterSpec(iv);

        byte[] key = new byte[16];
        Arrays.fill(key, (byte)1);
        SecretKeySpec keySpec = new SecretKeySpec(key, "AES");

        Cipher cipher = Cipher.getInstance("AES/CBC/NoPadding");

        long startE = System.currentTimeMillis();
        cipher.init(Cipher.ENCRYPT_MODE, keySpec, ivSpec);
        int len = cipher.update(plaintext, 0, plaintext.length, buf);
        len += cipher.doFinal(buf, len);
        long stopE = System.currentTimeMillis();

        byte[] ciphertext = Arrays.copyOf(buf, len-16);
        byte[] mac = Arrays.copyOfRange(buf, len-16, len);

        System.out.printf("AES/CBC of %dMB 0x00%n", MB);
        System.out.printf("Plaintext:  %s[MD5]%n",
            hexString(MessageDigest.getInstance("MD5").digest(plaintext)));
        System.out.printf("MAC:        %s%n", hexString(mac));

        long startD = System.currentTimeMillis();
        cipher.init(Cipher.DECRYPT_MODE, keySpec, ivSpec);
        int len2 = cipher.update(ciphertext, 0, ciphertext.length, buf);
        len2 += cipher.update(mac, 0, mac.length, buf, len2);
        len2 += cipher.doFinal(buf, len2);
        long stopD = System.currentTimeMillis();

        byte[] plaintext2 = Arrays.copyOf(buf, len2);
        System.out.printf("Decrypted:  %s[MD5]%n",
            hexString(MessageDigest.getInstance("MD5").digest(plaintext2)));

        System.out.printf(
            "Time used: encryption %d ms, decryption %d ms%n",
            stopE-startE, stopD-startD);
        System.out.printf(
            "Performance: encryption %.2f MB/s, decryption %.2f MB/s%n",
            MB*1000.0/(stopE-startE), MB*1000.0/(stopD-startD));
    }

}
```

Output

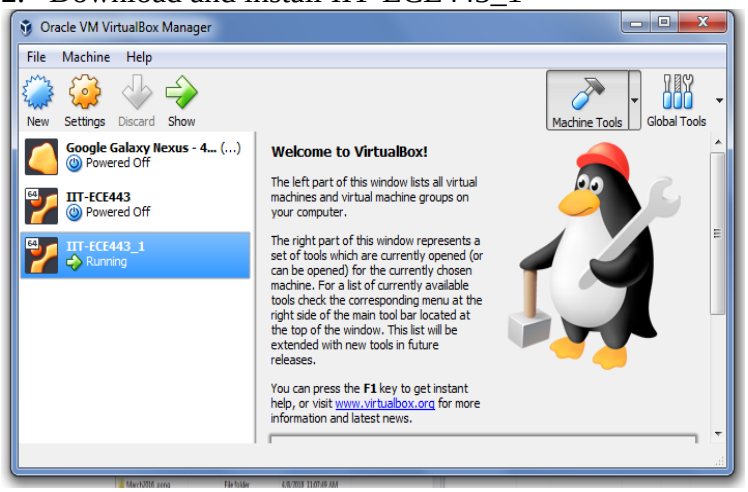




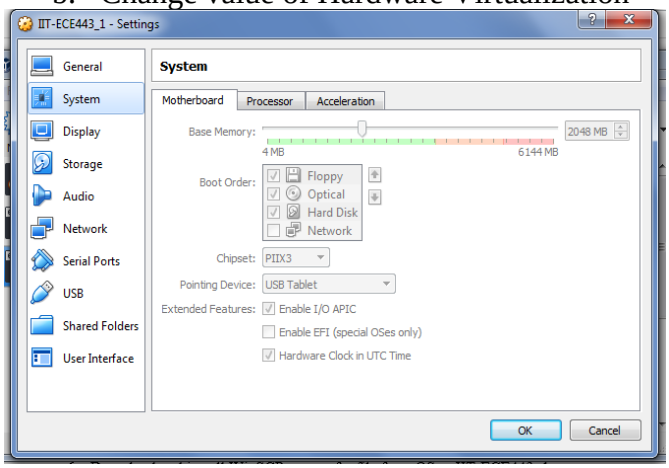
2. **A project report, which should include a summary of your experimental setup and a discussion of your findings.**

Procedure:

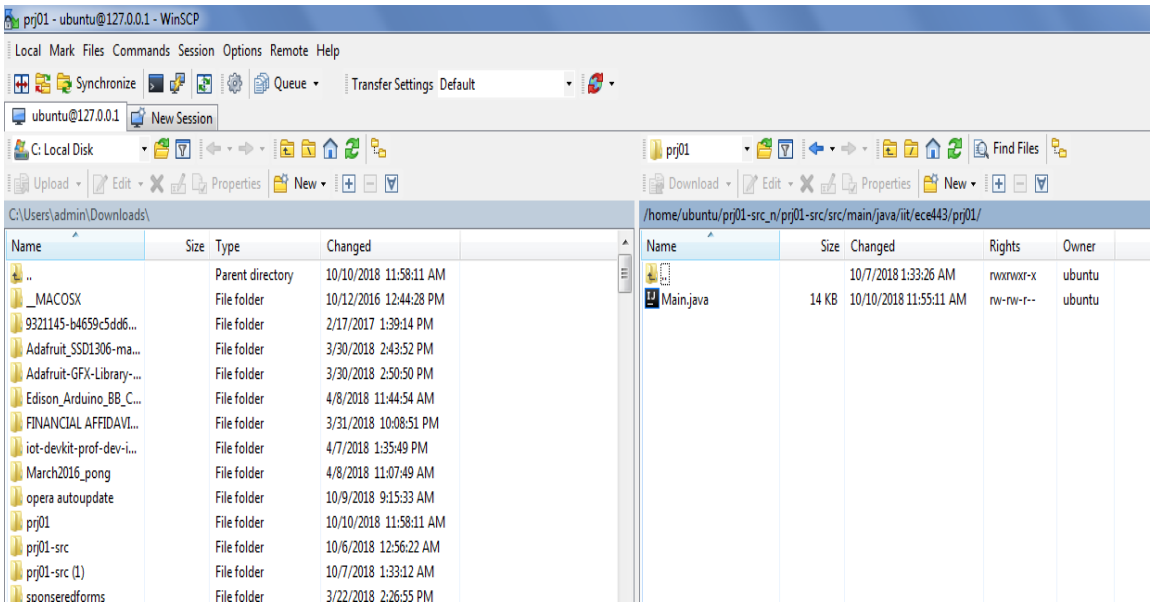
1. Install Oracle VM Virtualbox
2. Download and install IIT-ECE443_1



3. Change Settings in VM
    a. Motherboard : Set 2048 MB from 6144 MB
    b. Change value of Hardware Virtualization



4. Download and install Putty
5. Download prj01-src file
6. Download and install WinSCP to transfer file from OS to IIT-ECE443_1



7. Check and verify the Code of prj01-src and change the code according to Hash256, Hash512 for “Hello world!”.
8. I have searched for AES-GCM/CBC and research on that and run the program.
9. There should be change in bytes for CBC mode like from 12 bytes to 16 bytes.
10. Change IvParameter variables.
11. For Attack change the cipher text and when you try to decrypt, it will not possible.

Details of the processor and the OS (if you use a VM, both the host OS and the VM OS).

Details of the OS:

Windows 7 Ultimate
Processor: Intel(R) Core(TM) i5-2450M CPU @2.50 GHz
Memory (RAM): 6 GB (5.89 GB Usable)
System Type: 64-bit

Details of the VM:

IIT-ECE443_1
Processor: Intel(R) Core(TM) i5-2450M CPU @2.50 GHz
Type: Linux
Base Memory: 2.048GB
Version: Ubuntu (64-bit)

How can you be sure that you are using those APIs in correct ways?

From application and research with definitions and understanding of its usage , I can say I have used API in correct ways. I have given description of each given below.

import java.security.MessageDigest;

This MessageDigest class provides applications the functionality of a message digest algorithm, such as SHA-1 or SHA-256. Message digests are secure one-way hash functions that take arbitrary-sized data and output a fixed-length hash value.
A MessageDigest object starts out initialized. The data is processed through it using the update methods. At any point reset can be called to reset the digest. Once all the data to be updated has been updated, one of the digest methods should be called to complete the hash computation.The Java MessageDigest class represents a cryptographic hash function which can calculate a message digest from binary data. When you receive some encrypted data you cannot see from the data itself whether it was modified during transportation. A message digest can help alleviate that problem.To be able to detect if the encrypted data has been modified in transport, the sender can calculate a message digest from the data and send that along with the data. When you receive the encrypted data and message digest you can recalculate the message digest from the data and check if the message calculated digest matches the message digest received with the data. If the two message digests match there is a probability that the encrypted data was not modified during transport.
The digest method can be called once for a given number of updates. After digest has been called, the MessageDigest object is reset to its initialized state.

import java.util.Arrays;

The Arrays class of the java.util package contains several static methods that we can use to fill, sort, search, etc in arrays. This class is a member of the Java Collections Framework and is present in java.util.arrays.

import java.io.UnsupportedEncodingException

Java IO package defines java.io.UnsupportedEncodingException as a checked exception for catching it if programmers want to use the charset operation. If the JVM not supports the charset, then you will get the java.io.UnsupportedEncodingException thrown from your application.

import java.security.InvalidAlgorithmParameterException

This is the exception for invalid or inappropriate algorithm parameters.

import java.security.InvalidKeyException

This is the exception for invalid Keys (invalid encoding, wrong length, uninitialized, etc).

import javax.crypto.BadPaddingException

This exception is thrown when a Cipher operating in an AEAD mode (such as GCM/CCM) is unable to verify the supplied authentication tag.

import javax.crypto.IllegalBlockSizeException

You are ONLY ABLE to encrypt data in blocks of 128 bits or 16 bytes. That's why you are getting that IllegalBlockSizeException exception.In order to solve the problem we either have to pad our messages or we need to change to a non-padded cipher.For instance, we could use CBC mode (a mode of operation that effectively transforms a block cipher into a stream cipher) by specifying "AES/CBC/NoPadding" as the algorithm. Or PKCS5 padding by specifying AES/ECB/PKCS5Padding", which will automatically add some bytes at the end of your data in a very specific format to make the size of the ciphertext multiple of 16 bytes, and in a way that the decryption algorithm will understand that it has to ignore some data.

import javax.crypto.ShortBufferException

This exception is thrown when an output buffer provided by the user is too short to hold the operation result.

import java.security.NoSuchAlgorithmException

This exception is thrown when a particular cryptographic algorithm is requested but is not available in the environment.

import javax.crypto.AEADBadTagException

This exception is thrown when a Cipher operating in an AEAD mode (such as GCM/CCM) is unable to verify the supplied authentication tag.

import javax.crypto.KeyGenerator

This class provides the functionality of a secret (symmetric) key generator.

Key generators are constructed using one of the `getInstance` class methods of this class.

KeyGenerator objects are reusable, i.e., after a key has been generated, the same KeyGenerator object can be re-used to generate further keys.

There are two ways to generate a key: in an algorithm-independent manner, and in an algorithm-specific manner. The only difference between the two is the initialization of the object:

- **Algorithm-Independent Initialization**

  All key generators share the concepts of a *keysize* and a *source of randomness*. There is an init method in this KeyGenerator class that takes these two universally shared types of arguments. There is also one that takes just a keysize argument, and uses the SecureRandom implementation of the highest-priority installed provider as the source of randomness (or a system-provided source of randomness if none of the installed providers supply a SecureRandom implementation), and one that takes just a source of randomness.

  Since no other parameters are specified when you call the above algorithm-independent init methods, it is up to the provider what to do about the algorithm-specific parameters (if any) to be associated with each of the keys.

- **Algorithm-Specific Initialization**

  For situations where a set of algorithm-specific parameters already exists, there are two init methods that have an AlgorithmParameterSpec argument. One also has a SecureRandom argument, while the other uses the SecureRandom implementation of the highest-priority installed provider as the source of randomness (or a system-provided source of randomness if none of the installed providers supply a SecureRandom implementation).

import javax.crypto.NoSuchPaddingException

This exception is thrown when a particular padding mechanism is requested but is not available in the environment.

import javax.crypto.SecretKey;

A secret (symmetric) key.This interface contains no methods or constants. Its only purpose is to group (and provide type safety for) secret keys. Provider implementations of this interface must overwrite the equals and hashCode methods inherited from java.lang. Object, so that secret keys are compared based on their underlying key material and not based on reference. Keys that implement this interface return the string RAW as their encoding format (see getFormat), and return the raw key bytes as the result of a getEncoded method call. (The getFormat and getEncoded methods are inherited from the java.security.Key parent interface.)

import javax.crypto.Cipher

The Java Cipher (javax.crypto.Cipher) class represents an encryption algorithm. The term Cipher is standard term for an encryption algorithm in the world of cryptography. That is why the Java class is called Cipher and not e.g. Encrypter / Decrypter or something else.

import javax.crypto.spec.GCMParameterSpec

Specifies the set of parameters required by a Cipher using the Galois/Counter Mode (GCM) mode.

Simple block cipher modes (such as CBC) generally require only an initialization vector (such as IvParameterSpec), but GCM needs these parameters:

IV: Initialization Vector (IV)

tLen: length (in bits) of authentication tag T

In addition to the parameters described here, other GCM inputs/output (Additional Authenticated Data (AAD), Keys, block ciphers, plain/ciphertext and authentication tags) are handled in the Cipher class.

Please see RFC 5116 for more information on the Authenticated Encryption with Associated Data (AEAD) algorithm, and NIST Special Publication 800-38D, "NIST Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC."

The GCM specification states that tLen may only have the values {128, 120, 112, 104, 96}, or {64, 32} for certain applications. Other values can be specified for this class, but not all CSP implementations will support them.

import javax.crypto.spec.IvParameterSpec

This class specifies an initialization vector (IV). Examples which use IVs are ciphers in feedback mode, e.g., DES in CBC mode and RSA ciphers with OAEP encoding operation.

import javax.crypto.spec.SecretKeySpec

This class specifies a secret key in a provider-independent fashion.

It can be used to construct a SecretKey from a byte array, without having to go through a (provider-based) SecretKeyFactory.

This class is only useful for raw secret keys that can be represented as a byte array and have no key parameters associated with them, e.g., DES or Triple DES keys.

import javax.xml.bind.DatatypeConverter

The javaType binding declaration can be used to customize the binding of an XML schema datatype to a Java datatype. Customizations can involve writing a parse and print method for parsing and printing lexical representations of a XML schema datatype respectively.

Do the performance measurements meet your expectations? In case of AES with GCM, is it accelerated by hardware, e.g. via AES-NI and other special instructions?

AES-GCM is a NIST standardised authenticated encryption algorithm (FIPS 800-38D). Unfortunately the AES-GCM implementation used in Firefox (provided by NSS) until now did not take advantage of full hardware acceleration on all platforms; it used a slower software-only implementation on Mac, Linux 32-bit, or any device that doesn't have all of the AVX, PCLMUL, and AES-NI hardware instructions.

The algorithms were implemented in a uniform language (Java), using their standard specifications, and were tested on two different hardware platforms, to compare their performance. It shows that it depends on clock speed, processor and memory. When the clock speed is high, performance is good but if clock speed is low performance is not good.

The GCM cipher is using only 50 to 70% of the available CPU. It's also clear that the GCM cipher is doing more—specifically, providing a great deal more throughput than the CBC cipher—with significantly less compute power.

The system is 64-bit window7 with Intel core i5 2.5 GHz and 6GB RAM.

- SHA-256 is faster with 31% than SHA-512 only when hashing small strings. When the string is longer SHA-512 is faster with 28.9%.
    - EX.((2770-1977)/2770) *100 =28.9%
- MD5 produces 32 chars hash
- SHA-256 produces 64 chars hash
- SHA-512 produces 128 chars hash
- AES-NI also speeds up GCM by 4.2 to 8.5 times. Without AES-NI, CBC is faster than GCM in all packet sizes. With AES-NI, GCM almost takes back the crown of raw speed except the "16 bytes" category.