INTRODUCTION TO CRYPTOGRAPHY

Software Security

CHAPTER 16: EXERCISES FOR PART 3

By Group 4

Submitted to: Mr. Kelemwork

Acknowledgement

We Group four team members would like to express our appreciation for each other for the experience we shared while making this assignment. We are also thankful to Mr. Kelemwork, our Software Security teacher, for giving us with this chance and helping us complete it. The project enables us to know how to import packages, some details about crypto packages and more.

Group Members

Name	ID
Abrham Abayneh	1311576
Abrham Merkuz	1311579
Betelhem Negash	1306870
Bezawit Etsubneh	1306205
Hailemichael Mulugeta	1311614
Samrawit Fikremariam	1308042

Conceptual Exercises

1. List three advantages/disadvantages of using a web of trust model vs. using a certificate authority–based trust model.

Before diving to the question let us see what it means by PKI (public key infrastructure).

What is PKI?

Public Key Infrastructure (PKI) is a security framework that enables secure communication, data integrity, and digital signatures. It allows users, servers, and other devices to securely exchange information over open networks such as the Internet. PKI consists of a public key and a private key. The public key is used to encrypt data and then only the person with the “private key” can decrypt and view it. This ensures high-level security because even if an attacker was able to intercept the encrypted data, they would not be able to read it without having access to the other user’s private key. Additionally, PKI can also be used for verifying and authenticating electronic communication or documents using digital signatures.

There are a number of PKI models based on: whether the certificates are issued from one or more central authorities; the trust and revocation mechanisms being employed by the system; the levels of authority and control over the issuance and management of certificates; and whether other parties outside the core organization have a role to play in establishing, issuing, or managing certificates in some way. These are:

1. Certification Authority (CA) Model
2. Hierarchical CA Model
3. Distributed CA Model
4. Bridge CA Model
5. Web of Trust Model
6. Hybrid PKI Model
7. Grid PKI Model
8. Decentralized PKI (DPKI) Model

We are asked about the advantages and disadvantages of using a web of trust model vs. using a

certificate authority–based trust model. But what are those in the first place?

What is Web of Trust model?

The Web of Trust model is a distributed trust system in which users interact to create their own trust relationships. It relies on users to freely share information about other users and verify the authenticity of this information. This system does not have a central body that manages authentication, instead relying on peer-to-peer validation from trusted peers.

There are two types of this model. This are direct and indirect trust model.

A Direct Web of Trust is a trust model in which users must directly trust any transaction that is performed by another user in order for the transaction to be successful. This model relies on these direct connections for all instances of trust, and does not allow for an indirect or third-party system of validation.

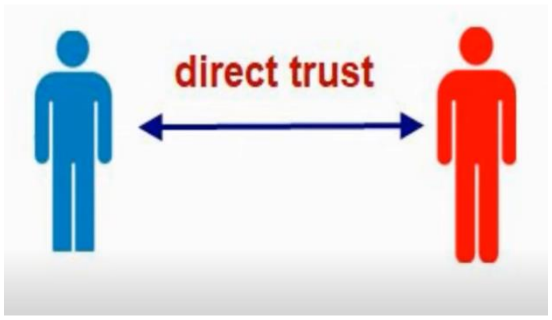


Figure 1: direct web of trust

An Indirect Web of Trust uses other people's or entities' ratings to grant permission for transactions to occur. In this type of system, users can form relationships through trust networks and benefit from other people's experiences and opinions about a particular vendor or user.

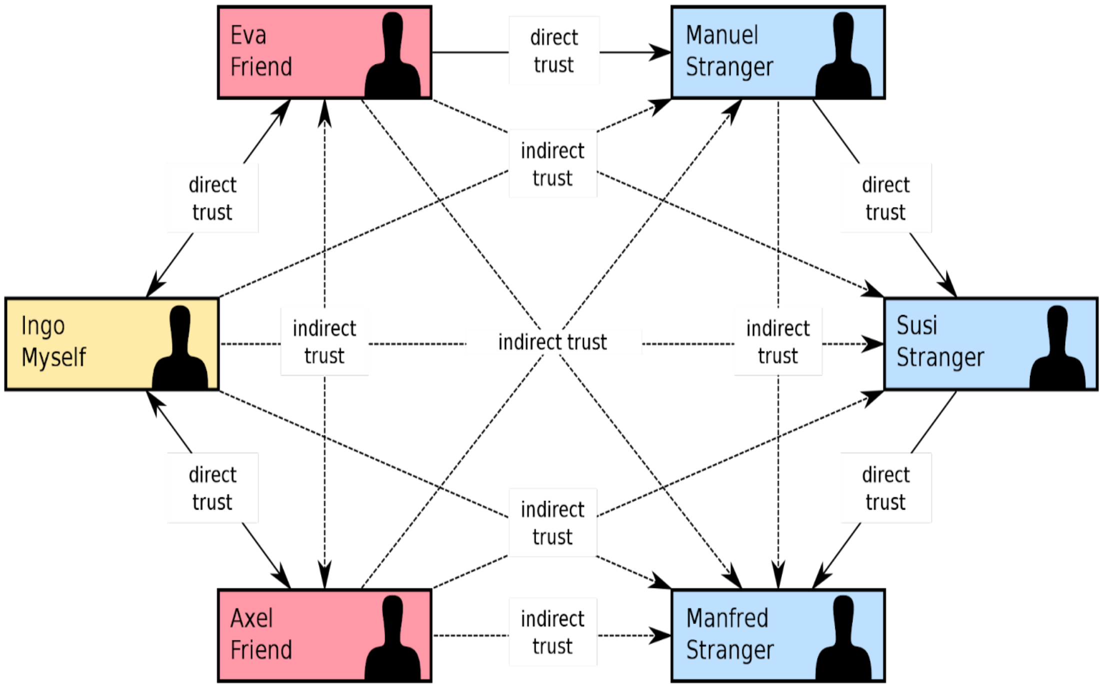


Figure 2: Indirect and direct web of trust

Advantages of web trust model

1. More flexibility as users may add or remove people from the network without having to contact a central authority.

2. Lower cost due to reduced infrastructure requirements and no need for ongoing payments to a centralized certificate authority.

3. People participate voluntarily, allowing them to decline if they don't feel comfortable with the security measures in place.

What are the disadvantages of web trust model?

1. Users must evaluate the validity of other users' certificates, making it difficult to guarantee correctness and accuracy in the verification process.

2. Security can be compromised by fraudsters or malicious actors who might get included into networks through false identities or forged credentials.

3. It is time-consuming, as each user has to perform steps manually that otherwise could have been completed automatically by touching a single button when working with trusted certificate authorities (CA).

What is certificate authority-based trust model?

A Certificate Authority-based Trust Model is a type of trust model that relies upon the assurance provided by digital certificates issued by an independent third-party Certificate Authority (CA). A CA is responsible for verifying the identity of certificate applicants, issuing digital certificates for those applicants and periodically validating their authenticity. Digital certificates can be used to facilitate secured communications between two parties without the need for manual validation. The trustworthiness of a Certificate Authority is based on its reputation, security protocols, audit requirements, and customer service.

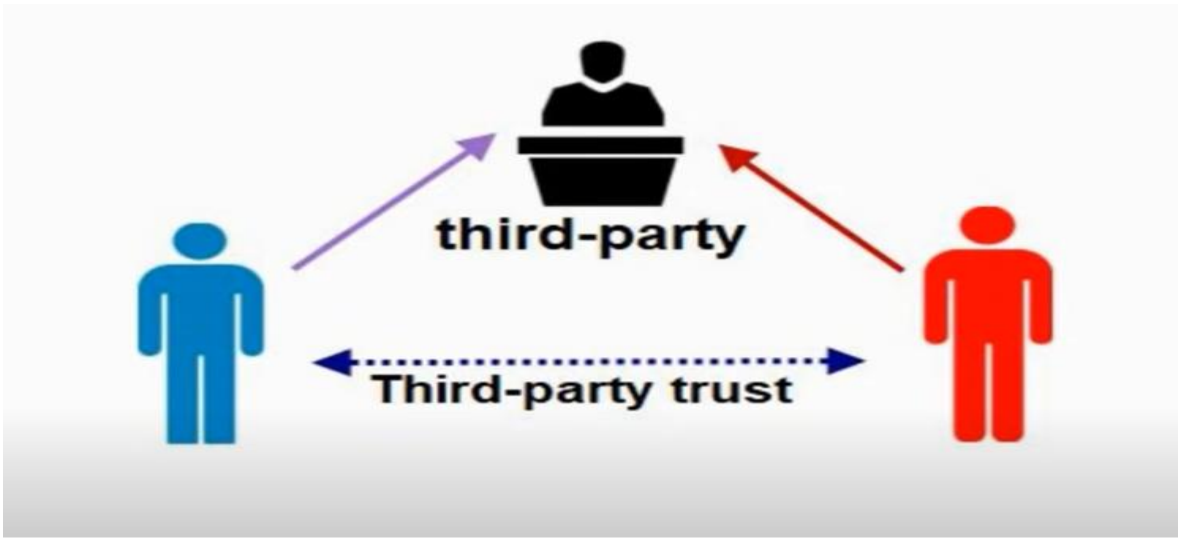


Figure 3: certificate authority-based model

Advantages of certificate authority-based trust model

1. Establishes trust beyond just individual hosts.

2. Establishes industry standard protocols for verified identities.

3. Ensures secure and efficient communication within the network.

4. Requires one time authentication, making it easier to manage authentication procedures in large networks.

5. Provides scalability and eliminates manual security checks of each host connection change.

6. Supports multiple cryptography algorithms as well as digital signature implementations to ensure data integrity and protect against malicious activities.

7. Reduced costs associated with identity verification processes due to its centralized nature and automated procedures for confirmation and authorization requests.

Disadvantages of certificate authority-based trust model?

1. Complex system which requires technical knowledge in order to properly implement and maintain the network's trust management systems securely across all participants on the network .

2. Reliance on a single central entity like Certificate Authority means that if it is compromised or fails, then the entire system would be impaired or ineffective as resulting trust may no longer be verifiable or accurate .

3. Restricted control for users over their preferred method of authentication since requests must go through an external third-party provider like a Certificate Authority, which can add additional latency in verifying identities before authenticity is established .

4. The initial cost of implementing such a system may be significant depending on the scale of the network it is applied to.

Web of trust model	Certificate authority-based model
- **Web of Trust requires a chain of signatures to be established from each user to the certificate in question – each user acting as his own Authority.**	- **An Authority is a person or organization that is assumed to be both well-known and trustworthy. If a chain of signatures can be established from an Authority to the certificate in question, then the trustworthiness of that certificate is assumed.**
- **A Web of Trust typically relies on personal knowledge – users are assumed to only have signed the certificates of those people they either know personally or have otherwise verified the identity of offline.**	- **The out of band confirmation of an Authority relies on the shrink wrap software model. "Out of band" means "face to face".**
- **Here each user is responsible for cultivating and maintaining his own outgoing signature chains. This means continually assessing the reliability of the downstream certificates, which even with the help of software tools requires a nontrivial amount of work. The reward for this effort is that the reliability of a well-maintained Web of Trust can be made arbitrarily high as more independent chains are established.**	- **Requires maintenance overhead, but Authorities are typically large organizations with well-paid staff, and the certificate chains are much shorter. Authorities also tend to use automated verification methods (such as emails) which can be used by an attacker to escalate from one form of compromise to another.**

2. State how you can use symmetric encryption to achieve (a) authentication, (b) confidentiality, and (c) message integrity.

Let us first discuss about symmetric encryption. It is a type of encryption where only one key is used to both encrypt and decrypt the data. It is also known as private-key encryption, because the same key is used for both operations. Unlike asymmetric encryption (also known as public-key encryption), symmetric encryption algorithms are much faster, which makes them more suitable for encrypting large amounts of data.

Now let us see how we can use symmetric encryption to achieve authentication, confidentiality and message integrity.

How can we achieve authentication using symmetric encryption?

Symmetric encryption, also referred to as secret key encryption, is an encryption scheme that uses the same key for both encrypting and decrypting data. It is a form of data security used to protect digital information by allowing only authorized users to access the encrypted information. It is one of the most widely used methods for authentication and can be used in a host of different applications. For example, symmetric encryption can be used to authenticate messages between two parties over the internet, or to protect sensitive documents such as credit card numbers.

In order to achieve authentication using symmetric encryption, firstly both parties (i.e., sender and receiver) must agree on a shared secret key and should be exchanged between them through secure means like through physically with each other or by transporting it via insecure channels with proper protection like SSL/TLS/VAULT. This is done so that the key is known only by both parties but not anyone else. Once the key has been agreed upon, either party can use this secret key to encrypt messages they wish to send, ensuring that only those who have agreed upon this same shared secret will be able to decipher it properly later.

When transmitting messages securely over the Internet using symmetric encryption, both parties must ensure that their communication channel is secure from outside interference or interception by malicious actors. To do this, each message consisting of plaintext must then be encrypted using a public-key cryptographic algorithm such as AES or Triple DES before being sent across the channel; after which it must then again be decrypted at its destination using a separate private-key algorithm. As long as no third-party individual knows either party's private keys (or indeed even accesses them), then this provides 'proof' of authenticity for each message transmitted – since only its intended recipient will receive it in an intelligible form suitable for further processing – thereby providing effective authentication.

How can we achieve confidentiality using symmetric encryption?

In symmetric-key encryption, a single key is used to encode (encrypt) and decode (decrypt) the data. This means that for the data to remain confidential it needs to be kept secret from viewers or attackers in order to maintain its security. To use symmetric encryption, two parties must first exchange keys through a secure channel. Both parties must have copies of the same key in order for encryption and decryption to occur correctly.

Once the key has been exchanged, both parties can use it simultaneously in different computer systems or devices both senders and receivers can then encrypt their messages using this shared key, which allows them to communicate securely while their messages remain confidential. In addition, they can even digitally sign their messages to ensure that tampering or frauds will not occur during the transmission process. Symmetric encryption also helps reduce computational resources since only one shared key needs to be managed rather than two distinct keys for separate users for each communication session.

An example of how symmetric encryption can be used to achieve confidentiality is when sending sensitive information over the internet. For example, if two people wanted to securely exchange financial information using email, they could do so using symmetric encryption. The data would be encrypted using the key and sent over a secure connection in order to maintain confidentiality.

How can we achieve message integrity using symmetric encryption?

Message integrity guarantee the validity of a message, ensuring that it has not been altered since it was sent by its original author. This can help protect against man-in-the-middle attacks and other malicious alterations of data. This can be achieved by combining two cryptographic primitives: Message Authentication Code (MAC) and Encryption.

A Message Authentication Code is a short block of code that is generated through a hash algorithm with a secret key as input. The MAC protects the integrity of the message, ensuring it hasn't been modified from its original form.

Encryption is the process of transforming plaintext into an unreadable form or ciphertext, which cannot be read until it has been decrypted with the proper secret key. When symmetric encryption is used, both the sender and recipient need to possess the same secret key in order to decipher the message.

By using symmetric encryption in combination with a MAC, both parties can verify that any messages they have exchanged have not been tampered with prior to arriving at its destination. Additionally, since only someone who possesses the same secret key can decrypt it, data confidentiality is also maintained.

We have already illustrated all this in the programming task that is given for us.

Programming Problem

3. Extend the AESEncrypter program of Section 12.1.6 to compute and verify a MAC on the message in addition to encrypting and decrypting data. Be sure to use different keys for the encryption and MAC computations

We looked at the AESEncrypter program from our book and attempted to decipher the code line by line. The book also explained how it works by using Chuck of codes, which allow us to expand the program to compute and validate MAC on the message.

We have seen what MAC is in question number 2.

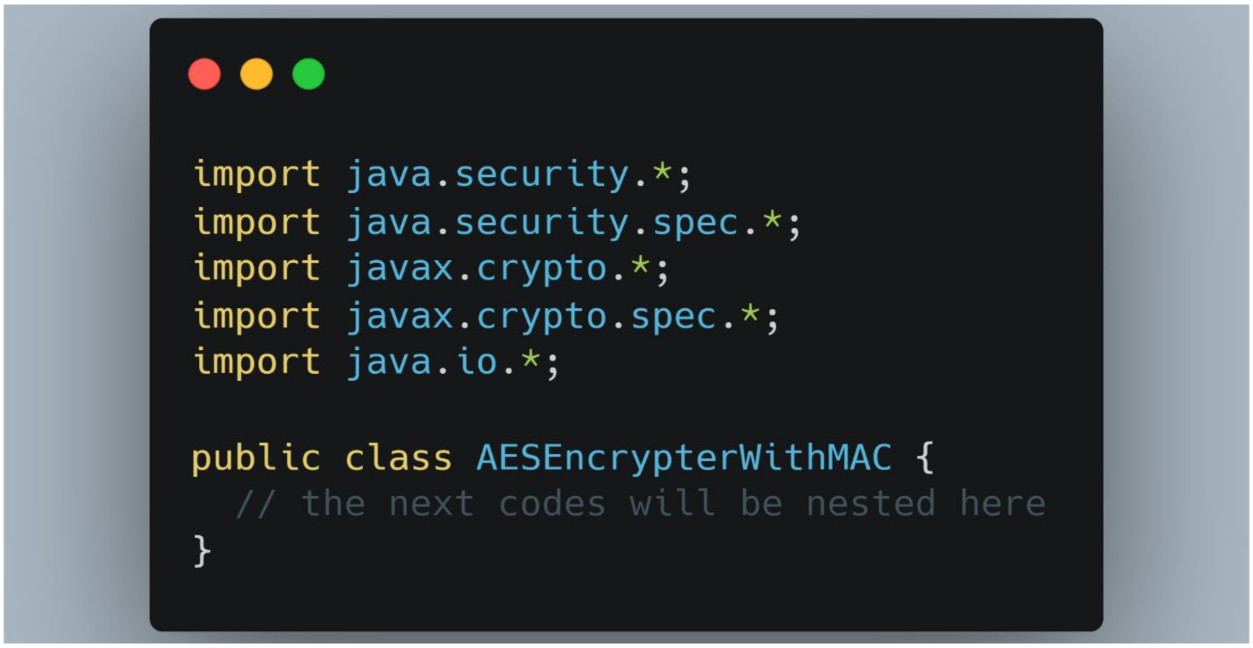
A Message Authentication Code (MAC) is a type of security algorithm used to authenticate data messages. It ensures the data message has been unaltered by its sender and prevents it from tampering by third parties. MACs are commonly used in communication protocols to verify both the integrity and authenticity of a transmitted message.

Example: In order for Alice to send her credit card details safely to Bob, who runs an e-commerce store, they could use a MAC algorithm to validate the transaction. Alice would begin by hashing her credit card information with a key known only to her, thus creating an authentication code that she transmits alongside the original data message. When Bob receives the message, he adds a hash using his own secret key and compares it to Alice's transmitted hash. If these two messages match, then Bob can be sure that no unauthorized third party has tampered with the message or attempted to view or alter its content during transmission without being detected.

Standing from the above example let's see how we implement MAC on a message.

The most common way to implement a MAC (Message Authentication Code) on a message while encrypting it is to use an HMAC (Hash-based Message Authentication Code). An HMAC is a cryptographic message digest generated with both a specific cryptographic hash algorithm and an encryption key. The HMAC can then be attached to the message before encryption, and optionally stored alongside the encrypted data.

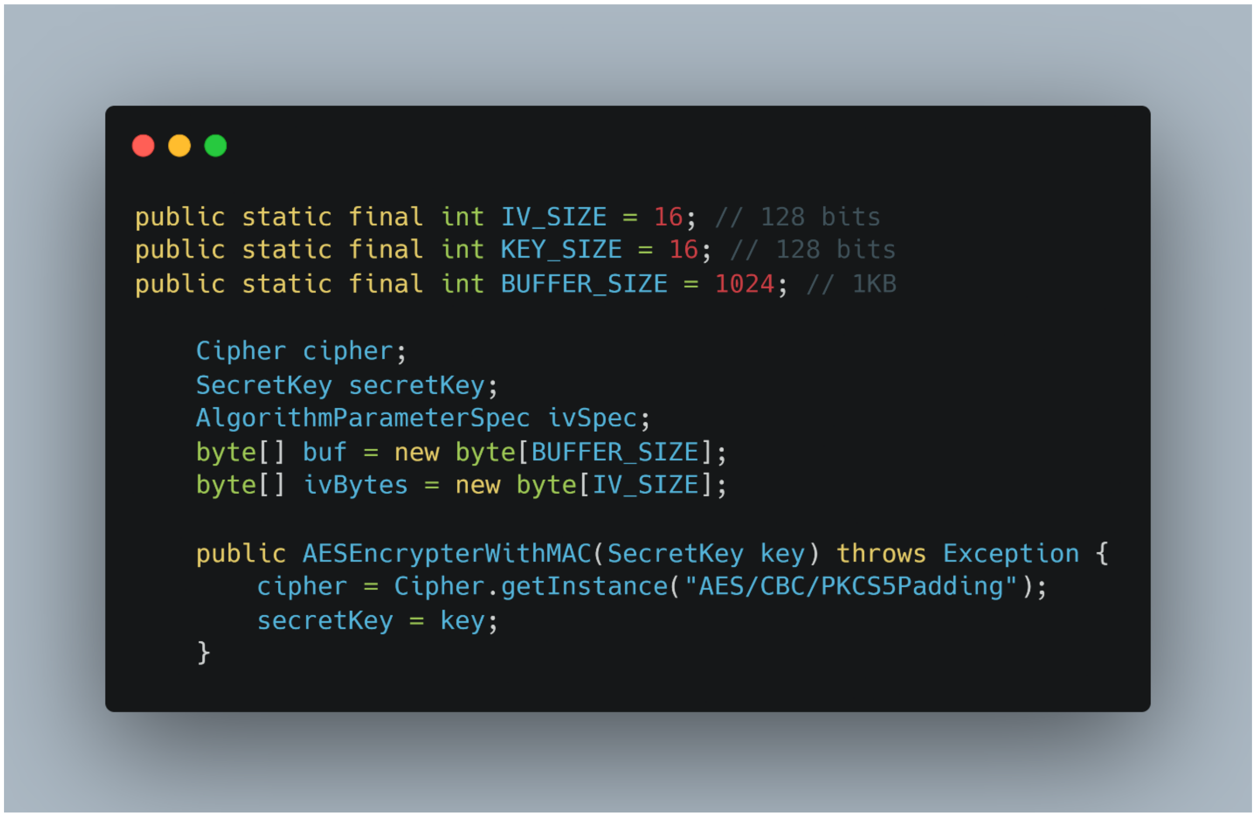
Code snippet 1



```java
import java.security.*;
import java.security.spec.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import java.io.*;

public class AESEncrypterWithMAC {
  // the next codes will be nested here
}
```

The above code is all about importing the necessary packages for the AES encryption. Next, we will see the components of this code that will be nested in the class that we created above.

Code snippet 2



```java
public static final int IV_SIZE = 16; // 128 bits
public static final int KEY_SIZE = 16; // 128 bits
public static final int BUFFER_SIZE = 1024; // 1KB

    Cipher cipher;
    SecretKey secretKey;
    AlgorithmParameterSpec ivSpec;
    byte[] buf = new byte[BUFFER_SIZE];
    byte[] ivBytes = new byte[IV_SIZE];

    public AESEncrypterWithMAC(SecretKey key) throws Exception {
        cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        secretKey = key;
    }
```

This is the definition of an AESEncrypter class that is used for encrypting different types of data using Advanced Encryption Standard (AES) algorithm in Cipher Block Chaining (CBC) mode.

The class provides constants to define the Initialization Vector (IV) size and Key size, as well as a buffer size which will be used to read data in blocks. It then defines a cipher object which will be used for encrypting the data.

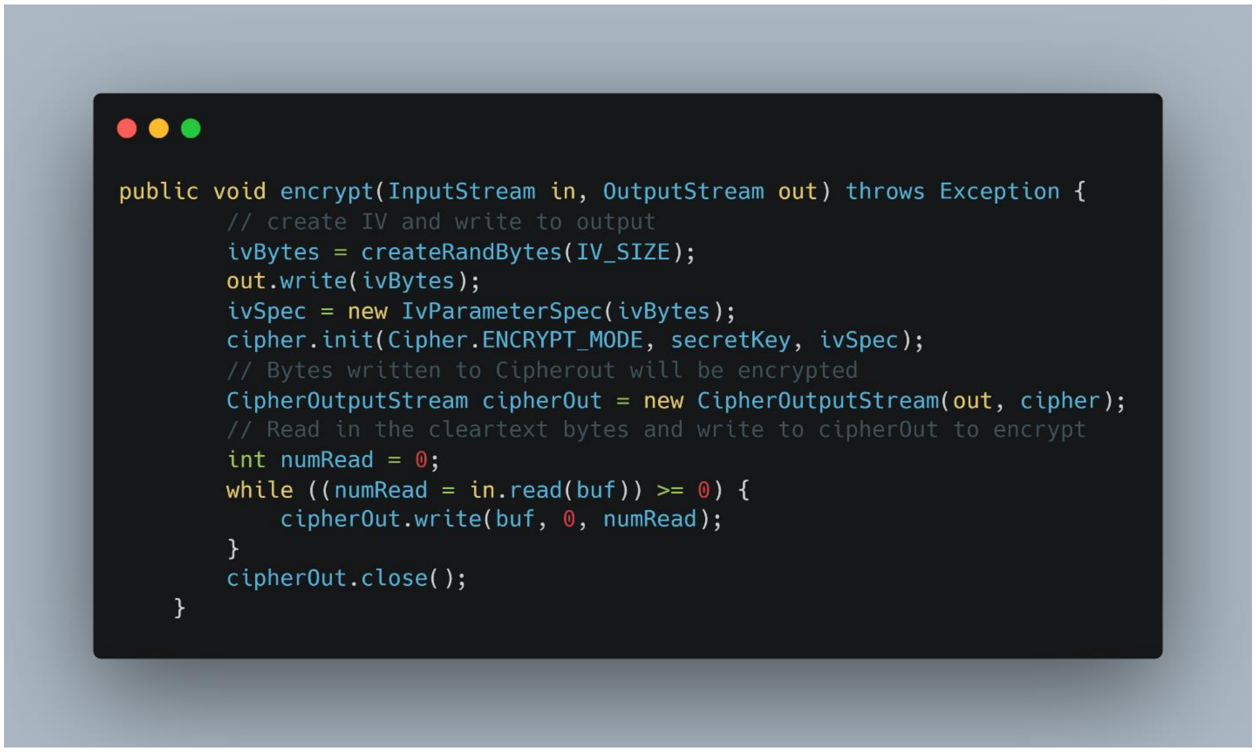
The secretKey variable is of type SecretKey and it stores an encryption key generated by a key generator.

The AlgorithmParameterSpec ivSpen stores parameters for initializing or reinitializing the cipher instance with its IV. This variable has to be initialized before using it for encryption/decryption operations.

The buf byte array is used to store data in memory a block at a time, while the ivBytes byte array is used to store the bytes of the initialized Initialization Vector.

The constructor takes in a parameter of type SecretKey, this is then stored in secretKey variable so it can be used later by the encryption process and then instantiates a Cipher object with AES/CBC/PKCS5Padding parameters indicating that we are using AES algorithm with Cipher Block Chaining mode and PKCS#5 padding scheme respectively.

Code snippet 3



```java
public void encrypt(InputStream in, OutputStream out) throws Exception {
        // create IV and write to output
        ivBytes = createRandBytes(IV_SIZE);
        out.write(ivBytes);
        ivSpec = new IvParameterSpec(ivBytes);
        cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivSpec);
        // Bytes written to Cipherout will be encrypted
        CipherOutputStream cipherOut = new CipherOutputStream(out, cipher);
        // Read in the cleartext bytes and write to cipherOut to encrypt
        int numRead = 0;
        while ((numRead = in.read(buf)) >= 0) {
            cipherOut.write(buf, 0, numRead);
        }
        cipherOut.close();
    }
```

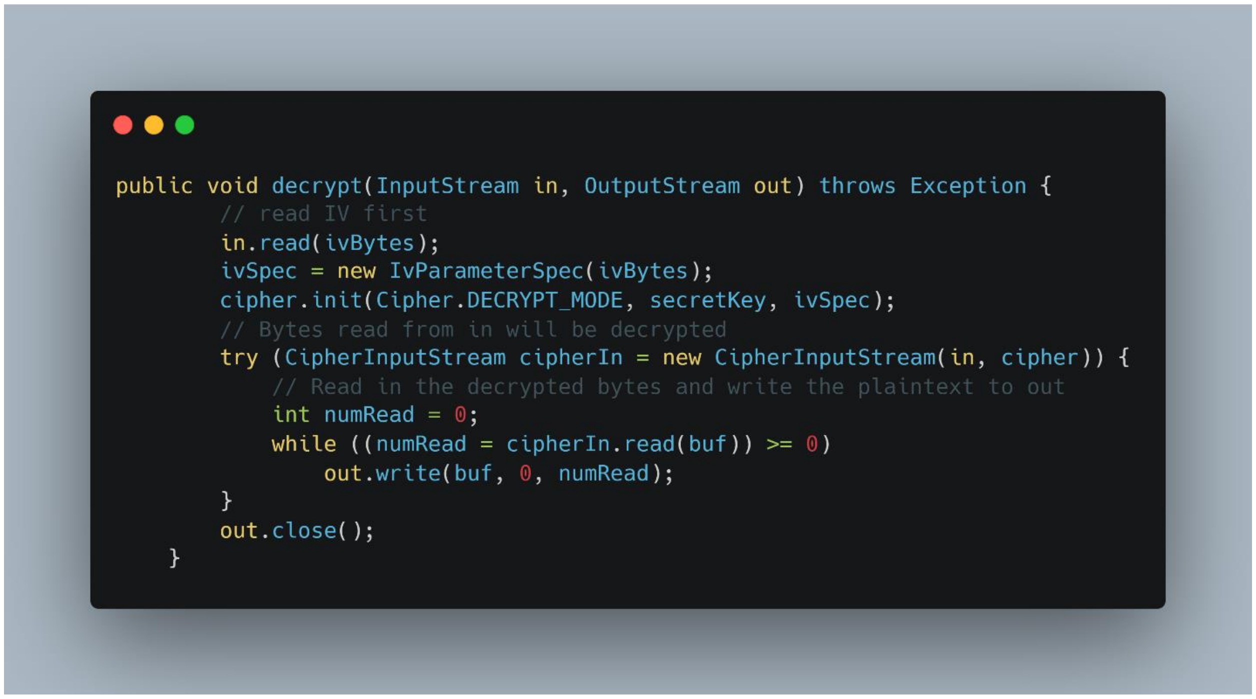
This code is an encryption method that takes input from an InputStream and writes it to the OutputStream after the data has been encrypted.

First, it creates a random initialization vector (ivBytes) and then writes it to the OutputStream.

An IvParameterSpec object is created using ivBytes and is then used to initialize the cipher in encryption mode and with a secret key.

A CipherOutputStream is then created using that cipher, which reads in clear text bytes from the InputStream, encrypts them, and then writes them out to the OutputStream. Once all of the data has been read from the InputStream and written to the CipherOutput Stream, it closes all of the streams.

Code snippet four

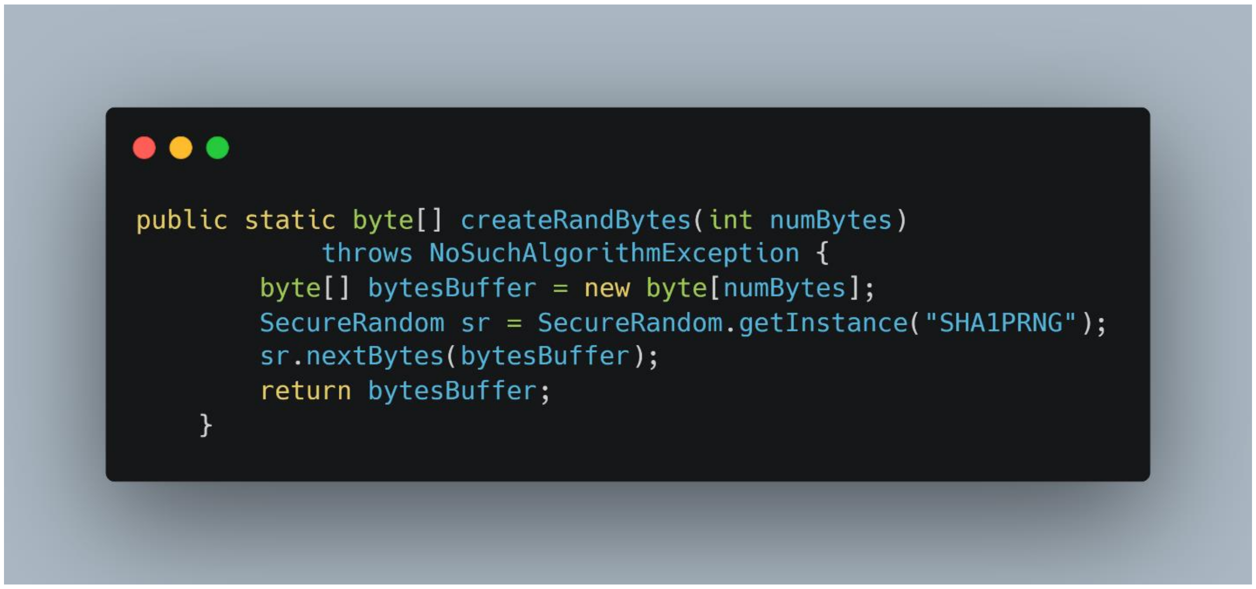


```java
public void decrypt(InputStream in, OutputStream out) throws Exception {
        // read IV first
        in.read(ivBytes);
        ivSpec = new IvParameterSpec(ivBytes);
        cipher.init(Cipher.DECRYPT_MODE, secretKey, ivSpec);
        // Bytes read from in will be decrypted
        try (CipherInputStream cipherIn = new CipherInputStream(in, cipher)) {
            // Read in the decrypted bytes and write the plaintext to out
            int numRead = 0;
            while ((numRead = cipherIn.read(buf)) >= 0)
                out.write(buf, 0, numRead);
        }
        out.close();
    }
```

This code is used to decrypt a stream of data that has been encrypted with the specified cipher. It first reads the initialization vector (IV) that was used for the encryption. This initialization vector is then used to initialize the Cipher, so that it can properly decrypt the data with the correct key.

After initializing the Cipher, a CipherInputStream is created which will begin decrypting data from the InputStream in and writing it to an OutputStream out. The decrypted data is then read from cipherIn, written to buf, and ultimately written out in plaintext to out. At the end, out is closed.

Code snippet five

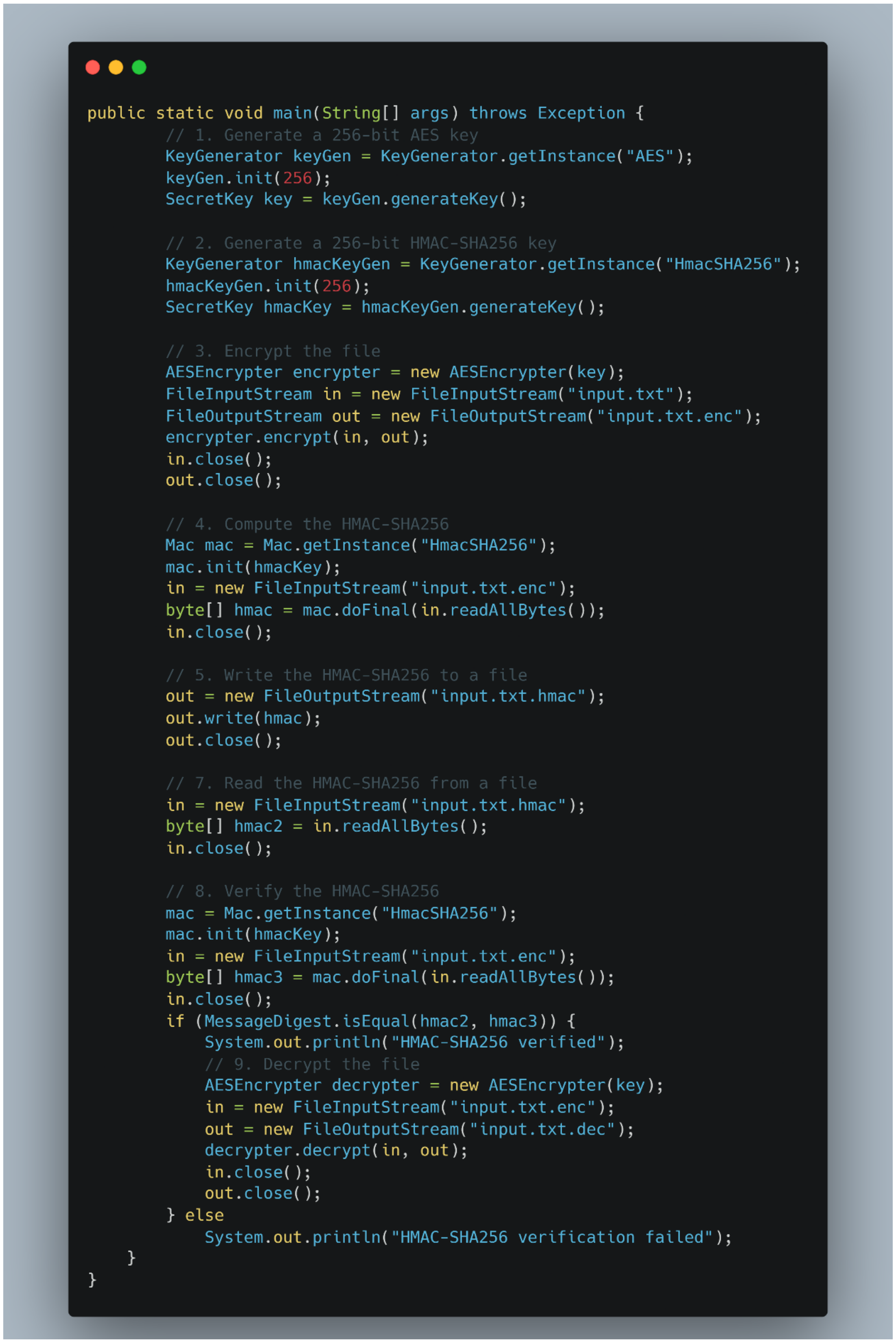


```java
public static byte[] createRandBytes(int numBytes)
            throws NoSuchAlgorithmException {
        byte[] bytesBuffer = new byte[numBytes];
        SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
        sr.nextBytes(bytesBuffer);
        return bytesBuffer;
    }
```

This code creates and returns an array of random bytes of a certain size. It does this by creating an array called "bytesBuffer" which is of length numBytes.

It then generates a secure random number using the SecureRandom class with the SHA1 algorithm, and assigns the randoms bytes to the byte array using the sr.nextBytes() method. Finally, it returns the array of random bytes.

Code snippet six



```java
public static void main(String[] args) throws Exception {
        // 1. Generate a 256-bit AES key
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(256);
        SecretKey key = keyGen.generateKey();

        // 2. Generate a 256-bit HMAC-SHA256 key
        KeyGenerator hmacKeyGen = KeyGenerator.getInstance("HmacSHA256");
        hmacKeyGen.init(256);
        SecretKey hmacKey = hmacKeyGen.generateKey();

        // 3. Encrypt the file
        AESEncrypter encrypter = new AESEncrypter(key);
        FileInputStream in = new FileInputStream("input.txt");
        FileOutputStream out = new FileOutputStream("input.txt.enc");
        encrypter.encrypt(in, out);
        in.close();
        out.close();

        // 4. Compute the HMAC-SHA256
        Mac mac = Mac.getInstance("HmacSHA256");
        mac.init(hmacKey);
        in = new FileInputStream("input.txt.enc");
        byte[] hmac = mac.doFinal(in.readAllBytes());
        in.close();

        // 5. Write the HMAC-SHA256 to a file
        out = new FileOutputStream("input.txt.hmac");
        out.write(hmac);
        out.close();

        // 7. Read the HMAC-SHA256 from a file
        in = new FileInputStream("input.txt.hmac");
        byte[] hmac2 = in.readAllBytes();
        in.close();

        // 8. Verify the HMAC-SHA256
        mac = Mac.getInstance("HmacSHA256");
        mac.init(hmacKey);
        in = new FileInputStream("input.txt.enc");
        byte[] hmac3 = mac.doFinal(in.readAllBytes());
        in.close();
        if (MessageDigest.isEqual(hmac2, hmac3)) {
            System.out.println("HMAC-SHA256 verified");
            // 9. Decrypt the file
            AESEncrypter decrypter = new AESEncrypter(key);
            in = new FileInputStream("input.txt.enc");
            out = new FileOutputStream("input.txt.dec");
            decrypter.decrypt(in, out);
            in.close();
            out.close();
        } else
            System.out.println("HMAC-SHA256 verification failed");
    }
}
```

The above code snippet is the main function of our AESEncrypter, that takes input(a message to be encrypted) from input.txt file. Then encrypt it by passing the in and out parameters to the encrypt function that we created earlier.

Then HMAC will be computed from the encrypted message, from input.txt.enc file, then it would be written to input.txt.hmac file.

Now the message is encrypted and HMAC is computed, but before decrypting the encrypted message we need to verify the HMAC. So, in order to verify it we need to read all the bytes from the input.txt.hmac file and compare it with the bytes that is computed from input.txt.enc

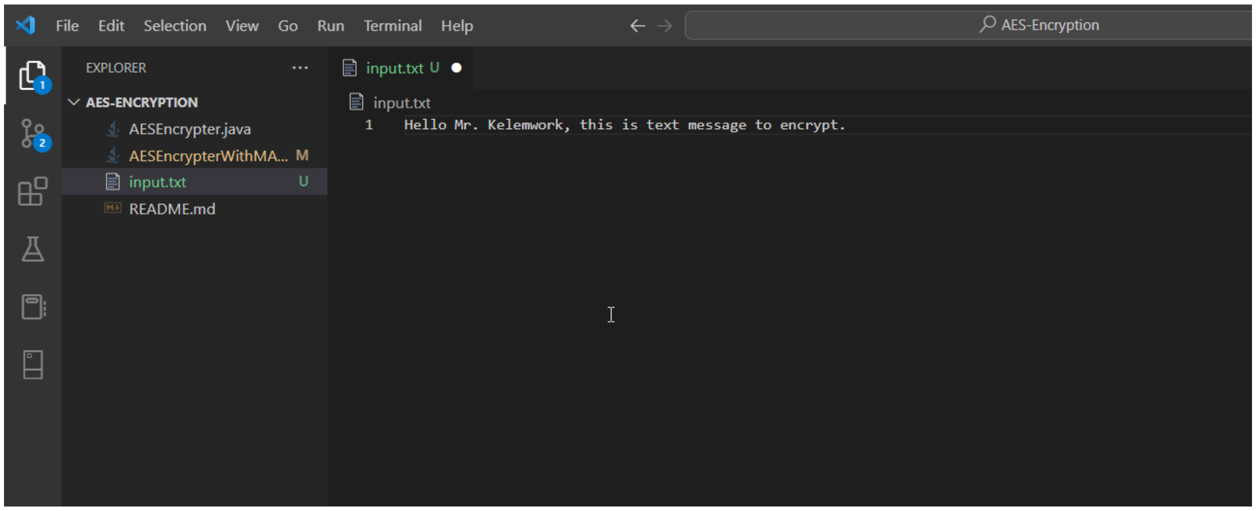
The 1st, 2nd and 3rd parts is all about generating 256-bit AES and HMAC-SHA256 cryptographic keys and then encrypting a file called "input.txt" using the AES encryption method. The encrypted file will be stored as "input.txt.enc".

Then on the 4th and 5th part, it sets up a Mac object to use the HmacSha256 algorithm. It then reads data from the encrypted file, input.txt.enc, to calculate a byte array HMAC-SHA256. Finally, it writes the byte array that is created to input.txt.hmac file.

On the last parts 6th, 7th, 8th and 9th part, it is reading a HMAC-SHA256 from an input.txt.hmac file and set it to hmac2 variable, initializing a hmacKey object with an HmacSHA256 instance, and creating an hmac3 variable with the doFinal method on an input.txt.enc file. It then checks if the HMAC-SHA256s in hmac2 and hmac3 match each other and will run the code to decrypt the file if they do match, or print that the verification failed if they do not match.

Let's check it now. Creating the input.txt file to provide the message for the encryption as we discussed earlier.

Input message

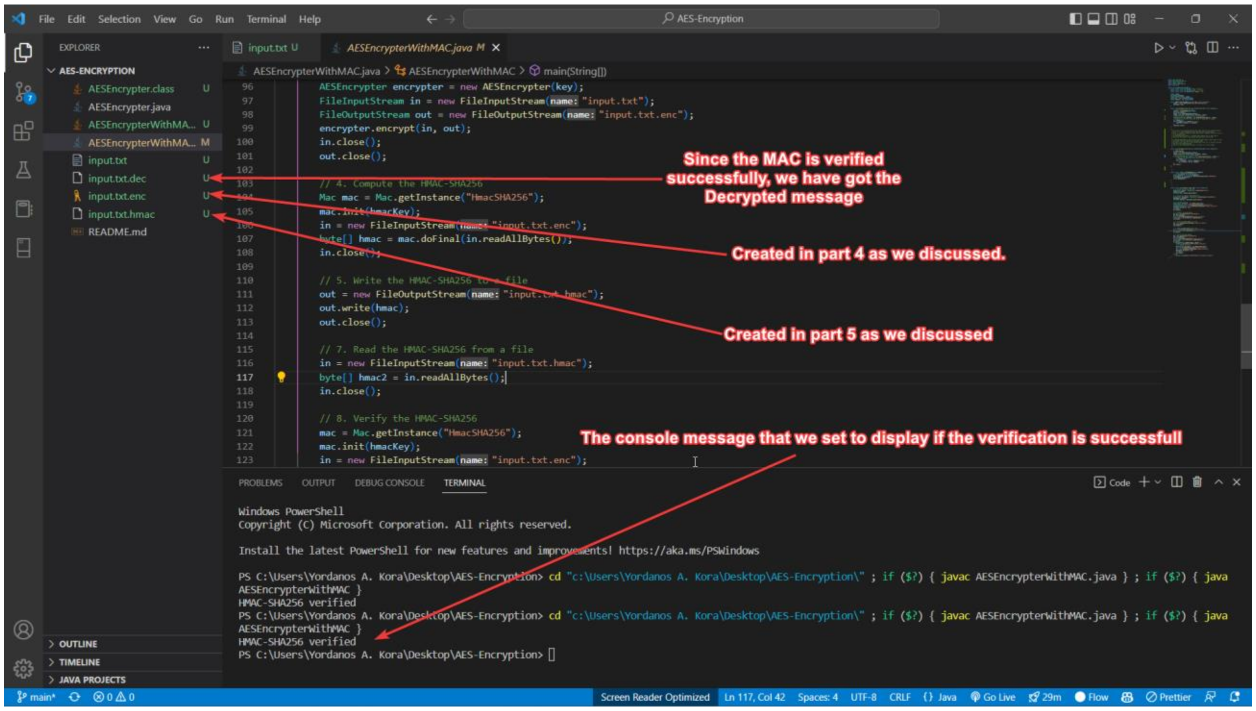


```
Hello Mr. Kelemwork, this is text message to encrypt.
```

POV: the AESEncruypter.java file is the code from the book without the MAC verification. And the AESEncrypterWithMAC.java file is with the MAC verification in it.

Let's run the AESEncrypterWithMAC.java and see what will happen.

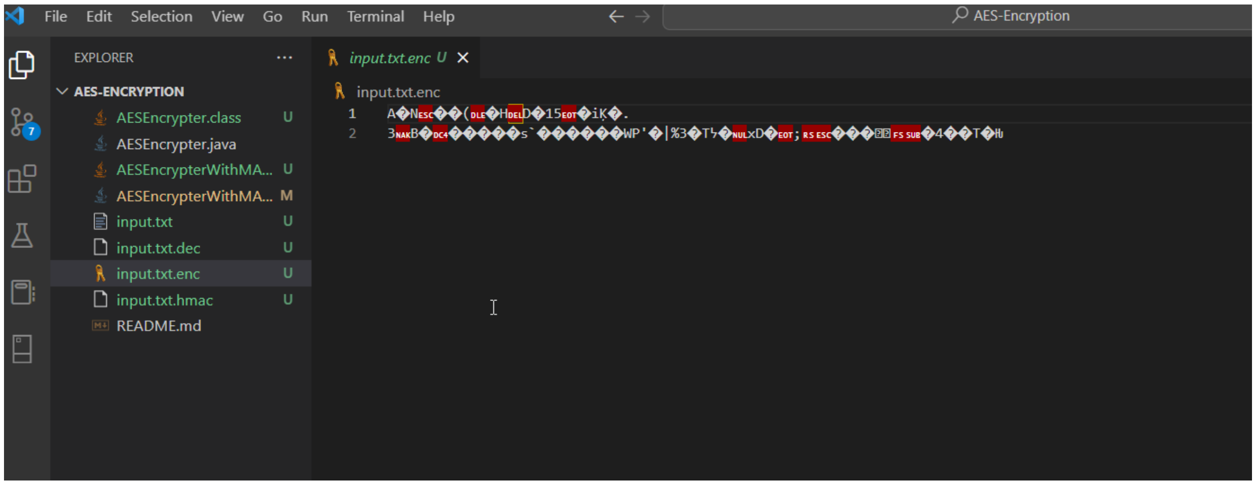
The whole result


Since the MAC is verified successfully, we have got the Decrypted message
Created in part 4 as we discussed.
Created in part 5 as we discussed
The console message that we set to display if the verification is successfull

```java
        AESEncrypter encrypter = new AESEncrypter(key);
        FileInputStream in = new FileInputStream(name: "input.txt");
        FileOutputStream out = new FileOutputStream(name: "input.txt.enc");
        encrypter.encrypt(in, out);
        in.close();
        out.close();

        // 4. Compute the HMAC-SHA256
        Mac mac = Mac.getInstance("HmacSHA256");
        mac.init(hmacKey);
        in = new FileInputStream(name: "input.txt.enc");
        byte[] hmac = mac.doFinal(in.readAllBytes());
        in.close();

        // 5. Write the HMAC-SHA256 to a file
        out = new FileOutputStream(name: "input.txt.hmac");
        out.write(hmac);
        out.close();

        // 7. Read the HMAC-SHA256 from a file
        in = new FileInputStream(name: "input.txt.hmac");
        byte[] hmac2 = in.readAllBytes();
        in.close();

        // 8. Verify the HMAC-SHA256
        mac = Mac.getInstance("HmacSHA256");
        mac.init(hmacKey);
        in = new FileInputStream(name: "input.txt.enc");
```

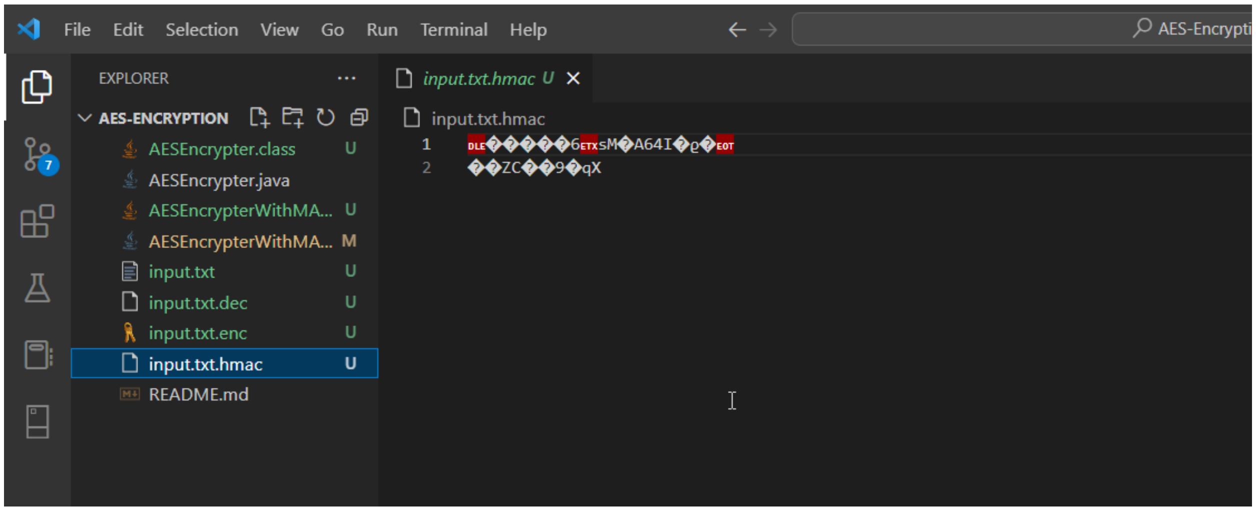
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Yordanos A. Kora\Desktop\AES-Encryption> cd "c:\Users\Yordanos A. Kora\Desktop\AES-Encryption\" ; if ($?) { javac AESEncrypterWithMAC.java } ; if ($?) { java AESEncrypterWithMAC }
HMAC-SHA256 verified
PS C:\Users\Yordanos A. Kora\Desktop\AES-Encryption> cd "c:\Users\Yordanos A. Kora\Desktop\AES-Encryption\" ; if ($?) { javac AESEncrypterWithMAC.java } ; if ($?) { java AESEncrypterWithMAC }
HMAC-SHA256 verified
PS C:\Users\Yordanos A. Kora\Desktop\AES-Encryption> 
```

Result for input.txt.enc



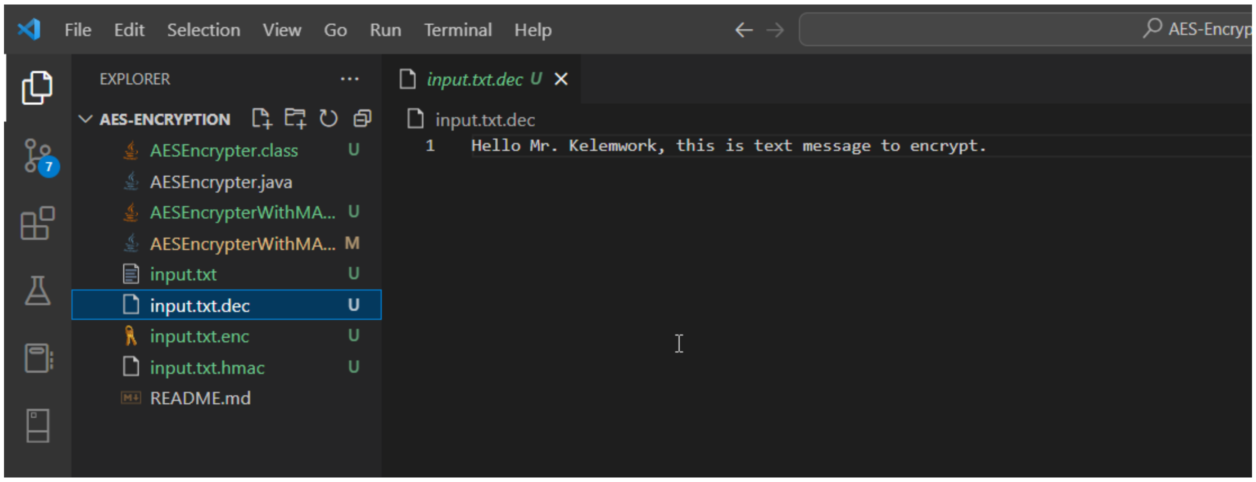
```
A�NESC��(DLE�HDELD�15EOT�iK�.
3NAKB�DC4������s`������WP'�|%3�T7�NULxD�EOT;RS ESC���FS SUB�4��T�Ht
```

Result for input.txt.hmac



```
DLE�����6ETXsM�A64I�ϱ�EOT
��ZC��9�qX
```

Result for input.txt.dec



```
Hello Mr. Kelemwork, this is text message to encrypt.
```

That’s all about how we extend the AESEncryper to comute and verify MAC on a message.