Instituto Politécnico Nacional

Escuela Superior de Cómputo

Session 5: Block ciphers

Cryptography

Group: 3CM6

Students:
Nicolás Sayago Abigail
Naranjo Ferrara Guillermo

Teacher:
Díaz Santiago Sandra

March 20, 2019

Contents

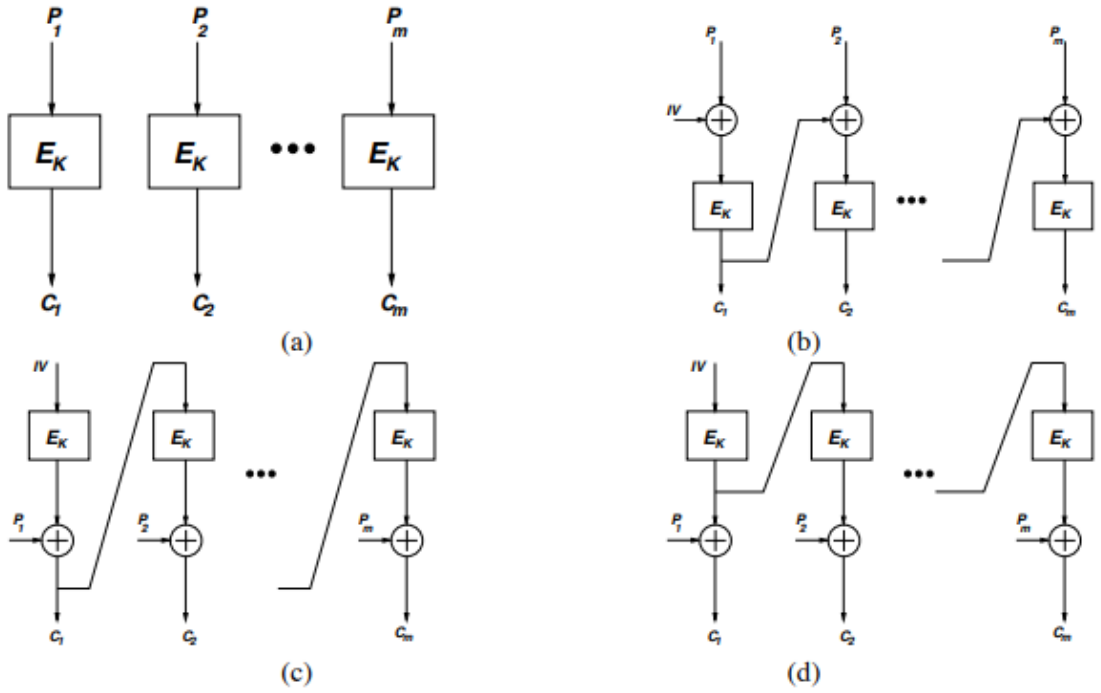
1 Block ciphers

Block ciphers are one of the most important primitives in cryptology. They are based on well-understood mathematical and cryptographic principles. Due to their inherent efficiency, these ciphers are used in many kinds of applications which require bulk encryption at high speed.

Formally a block cipher is considered to be secure if it behaves like a strong **pseudorandom permutation**, i.e., a block cipher is secure if an adversary cannot distinguish its output from a randomly chosen permutation. This definition of security for block ciphers is very strong, it implies that for any possible input, a secure block cipher should produce random outputs. It is unfortunate that there exists no formal security model for assessing whether a block cipher is or not secure or rather, how secure a cipher is.

2 Mode of operation

Block ciphers can only process plaintexts/ciphertexts with a bit length smaller than blocklength of the block cipher, which is typically less than 256 bits. But this is an unacceptable restriction since applications demand encryption/decryption of arbitrary long messages. In order to overcome this difficulty, it is necessary to introduce the concept of a mode of operation. The earliest modes of operation reported in the open literature were described back in 1981, in the standard FIPS Pub. 81 [17]. In that document four modes of operation were specified, namely, the Electronic Code Book (ECB), Cipher-Block Chaining (CBC), Cipher Feedback (CFB), and Output Feedback (OFB) modes, where the Data Encryption Standard (DES) was the underlying block cipher.

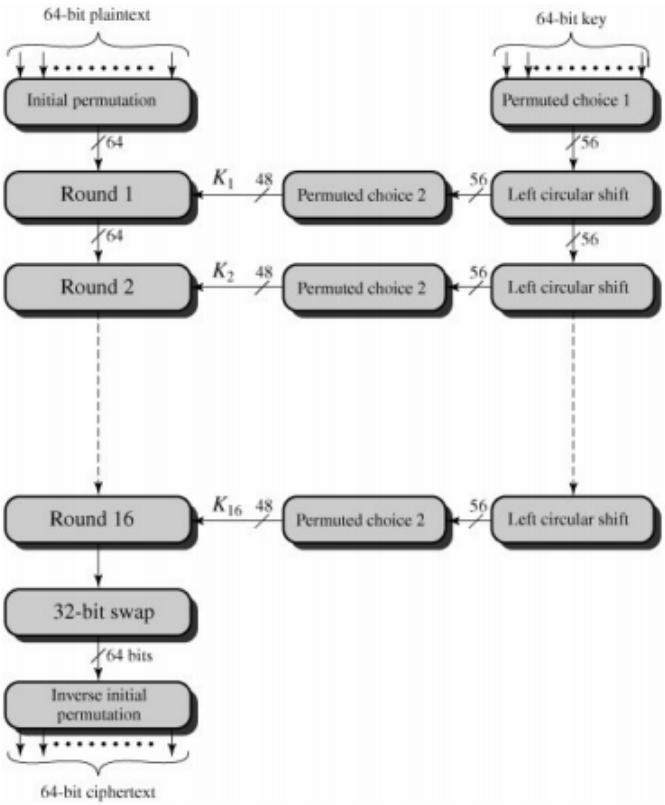


The second generation of modes of operation, therefore, was designed to offer other security services according to different application goals. Some of the most important classes of modes of operation are those modes which guarantee confidentiality, modes for authenticated encryption, modes for authenticated encryption with associated data and modes for disk encryption.

Another characteristic that distinguishes the second generation of modes of operation is the fact that they are mostly designed for operating with several or even arbitrary selections of block ciphers. The idea that a mode of operation is a research problem largely independent of the specific block cipher being used, may seem quite natural nowadays. Nevertheless, 25 years ago, when the first generation modes of operation were being specified, they were usually associated to an specific block cipher (typically DES or triple DES).

3 About Data Encryption Standard

For DES, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of stepes into a 64-bit output. The same stepes, with the same key, are used to reverse the encryption. Thre are two inputs to the encryption function: the plaintext to be encrypted and the key. In the next example, the plaintet must be 64 bits in length and key is 56 bits in length. (Actually, the function expects a 64-bit key as input. However, only 56 of these bits are ever used; the other 8 bits can be used as parity bits or simply set arbitrarily)



Looking at the left-hand side of the figure, we can see that the processing of the plaintext

proceeds in three phases.

- **Phase 1**
  The 64 bit plaintext passes through an initial permutation (**IP**) that rearranges the bits to produce the **permuted input**.
- **Phase 2**
  Consisting of 16 round of the same function, which involves voth permutation and substitution functions. The ourtput of the last (16) round consists of 64 bits that are a function of the input plaintext and the key. The left and rght halves of the output are swapped to produce the **preoutput**.
- **Phase 3**
  The preoutput is passed through a permutation(IP^{-1}) that is the inverse of the initial permutation function, to produce the 64-bit ciphertext. With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher.

4 Instructions

In this session we will use a cryptographic library to see how the blockciphers and the modes of operation work. To choose a cryptographic library, please only consider those that can be used with the programming languages: C, C++, Python, Java. To develop all the programming exercices you must use only one cryptographic library.

4.1 Programming exercises

Encipher and decipher files of different kind (.doc, .cpp, .java, .pdf, .jpeg, etc) using 3DES and different modes of operation. Consider the following requirements.

1. Using a function to generate cryptographically pseudorandom generator, provided by the cryptographic library, generate a random key and store it in a file, using base 6, i.e. only use the characters A-Za-z0-9,+/. Consider that with 3DES the key size can be 112 and 160 bits. Try both sizes.
2. To encipher, make proofs with the different alternatives we studied in class for 3DES: EEE and EDE with two and three keys. Also prove al the modes of operations CBC, CTR, OFB, CFB. Remember that the IV must be generated at random everytime you encipher a file. Generate the IV in the same way that you generate the key. Concatenate the IV with the ciiphertext and store them in a file.
3. Find out which are the mechanisms are included in your library for padding. Choose one of them whenever is required to encipher your files. Avoid t use only zeros for padding.
4. Decipher the files already encrypted.

5. Use file of different sizes (500kb, 1MB, 5MB, 10MB). Please measure the execution time and make a graph to compare the execution time versus the file size, to encipher with the differente versions of 3DES with the differente modes of operation.

5 Function used for EEE cipher with 2 keys

We have a function to cipher with EEE and each mode. In this function we received the following parameters:

- ✓ **String nameFile**: We used to know the file name which contain the plain text.
- ✓ **String mode**: Indicate which mode of operation will be used.
- ✓ **String nameFileKey**: Will be the file key name.
- ✓ **String nameFileIV**: Will be the file name that contains the IV and cipher text.

First we obtain the bytes from the file. (We have a function that makes it). Then we generate the keys. We are using **Key Generator** (We explain key generater in an subsection) to get a key, we can put the size of the key, that is 56. In this case, we only need 2 keys, so we have an array of keys, we used one key two times and then we have 3 keys to EEE.

We convert each key to string and then convert it in an array of bytes to put in a file. We can see this file when we run the program cause is generated.

We used Cipher function by JAVA, this class provides the functionality of a cryptographic cipher for encryption and decryption. We made a transformation, it is a string that describes the operation (or set of operations) to be performed on the given input, to produce some output. A transformation always includes the name of a cryptographic algorithm (e.g., DES), and may be followed by a feedback mode and padding scheme.

To cipher with EEE, we use DES tree times. We only have a restriction, that is when we cipher the first time, we need a padding scheme, DES algorithm requires that the input data to be 8-byte blocks. If you want to encrypt a text message that is not multiples of 8-byte blocks, the text message must be padded with additional bytes to make the text message to be multiples of 8-byte blocks.

For that reason, PKCS5Padding is a padding scheme described in: RSA Laboratories and JAVA allow used it.

For the IV vectors, we used a **secure Random** (We explain secure random in an subsection). We created an bytes array of secure Random to cipher. We use cipher init to cipher the plain text with the parameters: Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness. The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of opmode .

In this case we use ENCRYPTMODE. We get the IV created and we sent it to the new file that will be readed when we want decipher.

We save the final cipher text in a file with the IV array.

✓Code

```
public static void modeEEE2Keys(String nameFile, String mode, String
 ↪  nameFileKey, String nameFileIV) throws Exception {
    int i;
    byte[][] IV = new byte[3][];
    byte[] keysBytes, cipherText, decrypt;
    SecretKey[] desKey;
    String c, keyString, nameIV;

    // Get bytes of message
    byte[] message = createByte(nameFile);

    // Generate keys
    KeyGenerator keygenerator = KeyGenerator.getInstance("DES");
    keygenerator.init(56); // Size of key

    // Get keys, we use the same key for i = 2, 3
    desKey = new SecretKey[3];
    for(i = 0; i < 3; i++) {
      if(i == 2)
        desKey[i] = desKey[1];
      else
        desKey[i] = keygenerator.generateKey();
    }
    keyString = "";
    for(i = 0; i < 3; i++)
      keyString = keyString +
       ↪  Base64.getEncoder().encodeToString(desKey[i].getEncoded());

    keysBytes = keyString.getBytes();

    createFile(keysBytes, nameFileKey);

    cipherText = message;
    c = "DES/" + mode + "/PKCS5Padding";
    for(i = 0; i < 3; i++) {
      if(i > 0)
        c = "DES/" + mode + "/NoPadding";
      Cipher cipher = Cipher.getInstance(c);

      // Generate an initialization vector (IV)
      SecureRandom secureRandom = new SecureRandom();
```

```
      byte[] ivspec = new byte[cipher.getBlockSize()];
      secureRandom.nextBytes(ivspec);

      cipher.init(Cipher.ENCRYPT_MODE, desKey[i], secureRandom);
      IV[i] = cipher.getIV();
      cipherText = cipher.doFinal(cipherText);
    }
    createFileIVText(cipherText, IV, nameFileIV);
  }
```

5.1 Secure Random

This class provides a cryptographically strong random number generator (RNG). A cryptographically strong random number minimally complies with the statistical random number generator tests specified in FIPS 140-2, Security Requirements for Cryptographic Modules, section 4.9.1. Additionally, SecureRandom must produce non-deterministic output. Therefore any seed material passed to a SecureRandom object must be unpredictable, and all SecureRandom output sequences must be cryptographically strong, as described in RFC 1750: Randomness Recommendations for Security.

5.2 Key generator

This class from JAVA provides the functionality of a secret (symmetric) key generator. KeyGenerator objects are reusable, i.e., after a key has been generated, the same KeyGenerator object can be re-used to generate further key, (for this reason we only declared one KeyGenerater iven when we needed 2 keys).

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.

6 Functions used for EEE cipher with 3 keys

We have a function to cipher with EEE and each mode. In this function we received the following parameters:

- ✓ **String nameFile**: We used to know the file name which contain the plain text.
- ✓ **String mode**: Indicate which mode of operation will be used.
- ✓ **String nameFileKey**: Will be the file key name.
- ✓ **String nameFileIV**: Will be the file name that contains the IV and cipher text.

First we obtain the bytes from the file. (We have a function that makes it). Then we generate the keys. We are using **Key Generator** (We have already explained key generater in an subsection before) to get a key, we can put the size of the key, that is 56. In this case, we generate 3 keys, so we have an array of keys, that contains each one.

We convert each key to string and then convert it in an array of bytes to put in a file. We can see this file when we run the program cause is generated.

We used Cipher function by JAVA, this class provides the functionality of a cryptographic cipher for encryption and decryption. We made a transformation, it is a string that describes the operation (or set of operations) to be performed on the given input, to produce some output. A transformation always includes the name of a cryptographic algorithm (e.g., DES), and may be followed by a feedback mode and padding scheme.

To cipher with EEE, we use DES tree times. We only have a restriction, that is when we cipher the first time, we need a padding scheme, DES algorithm requires that the input data to be 8-byte blocks. If you want to encrypt a text message that is not multiples of 8-byte blocks, the text message must be padded with additional bytes to make the text message to be multiples of 8-byte blocks.

For that reason, PKCS5Padding is a padding scheme described in: RSA Laboratories and JAVA allow used it.

For the IV vectors, we used a **secure Random** (Wehave already explained secure random in an subsection). We created an bytes array of secure Random to cipher. We use cipher init to cipher the plain text with the parameters: Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness. The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of opmode .

In this case we use ENCRYPTMODE. We get the IV created and we sent it to the new file that will be readed when we want decipher.

We save the final cipher text in a file with the IV array.

✓Code

```
public static void modeEEE(String nameFile, String mode, String nameFileKey,
 ↪  String nameFileIV) throws Exception {
    // Get bytes of message
    int i;
    byte[][] IV = new byte[3][];
```

```
    byte[] keysBytes, cipherText;
    SecretKey[] desKey;
    String keyString;

    byte[] message = createByte(nameFile);

    // Use a key generator
    KeyGenerator keygenerator = KeyGenerator.getInstance("DES");
    keygenerator.init(56); // Size of key
    desKey = new SecretKey[3];
    // Generate a key
    for(i = 0; i < 3; i++)
      desKey[i] = keygenerator.generateKey();

    keyString = "";
    for(i = 0; i < 3; i++)
      keyString = keyString +
       ↪  Base64.getEncoder().encodeToString(desKey[i].getEncoded());

    keysBytes = keyString.getBytes();

    createFile(keysBytes, nameFileKey);

    cipherText = message;

    String c = "DES/"+ mode +"/PKCS5Padding";
    for(i = 0; i < 3; i++) {
      if(i > 0)
        c = "DES/"+ mode +"/NoPadding";
      Cipher cipher = Cipher.getInstance(c);
      // generate an initialization vector (IV)
      SecureRandom secureRandom = new SecureRandom();
      byte[] ivspec = new byte[cipher.getBlockSize()];
      secureRandom.nextBytes(ivspec);
      cipher.init(Cipher.ENCRYPT_MODE, desKey[i], secureRandom);
      IV[i] = cipher.getIV();
      cipherText = cipher.doFinal(cipherText);
    }
    createFileIVText(cipherText, IV, nameFileIV);
  }
```

7 Functions used to Decrypt EEE

We have a function to decipher with EEE and each mode. In this function we received the following parameters:

- ✓ **String mode**: Indicate which mode of operation will be used.
- ✓ **String nameFileKey**: We used to know the file name which contain the key.
- ✓ **String nameFileIV**:We used to know the file name which contain the Cipher text and the IV.

First we obtain the key from the file (We have a function that makes it). We convert each string, that represent a key, to a valid secret key. Then we obtain the cipher text from a file and the same thing to the IV array.

We decipher like we cipher. We use the function Cipher but now we have padding the last time we make the EEE, in this case we do not use CRYPT MODE, now we use DECRYPT MODE, and we can obtain the plain text.

Finally we save the plain text in a new file.

✓Code

```
public static byte[] modeEEEdecipher(String mode, String nameFileKey, String
 ↪  nameFileIV) throws Exception {
    int i, j;
    byte[][] IV;
    IV = new byte[3][];
    byte[] plainText, toDecrypt;
    SecretKey[] desKey;
    String aux = "";
    String c = "DES/"+ mode +"/NoPadding";

    desKey = new SecretKey[3];
    createByteKey(nameFileKey);

    // Get key from file
    aux = new String(originalKey_1);
    byte[] decodedKey = Base64.getDecoder().decode(aux);
    desKey[0] = new SecretKeySpec(decodedKey, 0, decodedKey.length, "DES");
     ↪  // decode the base64 encoded string
    aux = new String(originalKey_2);
    decodedKey = TBase64.getDecoder().decode(aux);
    desKey[1] = new SecretKeySpec(decodedKey, 0, decodedKey.length, "DES");
     ↪  // decode the base64 encoded string
```

```
    aux = new String(originalKey_3);
    decodedKey = Base64.getDecoder().decode(aux);
    desKey[2] = new SecretKeySpec(decodedKey, 0, decodedKey.length, "DES");
     ↪  // decode the base64 encoded string

    IV[0] = new byte[8];
    IV[1] = new byte[8];
    IV[2] = new byte[8];

    // Get bytes to decrypt
    toDecrypt = createByteIVText(nameFileIV);
    plainText = toDecrypt;
    IV[0] = iv_1;
    IV[1] = iv_2;
    IV[2] = iv_3;

    for(i = 1; i <= 3; i++) {
      if(i - 3 == 0)
        c = "DES/"+ mode +"/PKCS5Padding";

      Cipher decipher = Cipher.getInstance(c);
      decipher.init(Cipher.DECRYPT_MODE, desKey[3-i], new
       ↪  IvParameterSpec(IV[3-i]));
      plainText = decipher.doFinal(plainText);
    }
    return plainText;
  }
```

8 Functions used for EDE with 2 keys

8.1 Encrypt

We have a function to cipher with EDE. In this function we receive the following parameters:

- ✓ **byte[] message**: It contains the message to encrypt.
- ✓ **String outFileName**: The name of the output file.
- ✓ **String mode**: Indicate which mode of operation will be used.

First we set the strings that will be passed as parameters for the instances of the objects KeyGenerator and Cipher, depending on the mode of operation specified.

These strings set out the algorithm that we'll use (in this case is DES), the mode of operation which we'll be working, and the padding scheme to use (JCE only provides us PKCS5Padding as padding scheme or the option to not use it).

Then we create an array of type IvParamSpec, object that allows us to pass an IV as a parameter to an instance of Cipher, because of the use of DES three times, we'll need 3 IV's.

Now we do the first encryption, for what we generate a valid key for DES, which will be encoded in base 64, and after do de encryption, we obtain the IV used and stores it in our array previously mentioned.

After that we proceed with the decryption of the message generated in the step before but with the other key, but because there hasn't been an encryption with this key, we don't have an IV that we'll need in the decryption of the message, so we had to generate one. We did that the same as the key, but we only took the 8 first bytes in the key generated to act as our second IV, which also is stored in the array.

It's worth to mention that because in the first encryption was used PKCS5 padding, all of our bytes in the message are complete, and because we are using a different key to decrypt the message, we mustn't use padding in the decryption, if that were the case, there would be an error.

Finally we do the second encryption of the message generated in the step before with the first key, and, as in the first encryption, we get the IV used and stores it in the array.

The only thing left is to create the output files, one for the keys and other for the IV's and the ciphertext. After that we return an array of strings that contains the names of both output files.

✓Code

```
public static String[] EDEcipher_2Keys(byte[] message, String outFileName,
 ↪  String mode) throws NoSuchAlgorithmException, NoSuchPaddingException,
            InvalidKeyException, InvalidAlgorithmParameterException,
             ↪  IllegalBlockSizeException, BadPaddingException, IOException{

        String mc = "", md = "";
        switch(mode){
            case "cbc":
                mc = "DES/CBC/PKCS5Padding";
                md = "DES/CBC/NoPadding";
                break;
            case "ofb":
                mc = "DES/OFB/PKCS5Padding";
                md = "DES/OFB/NoPadding";
                break;
            case "cfb":
                mc = "DES/CFB/PKCS5Padding";
                md = "DES/CFB/NoPadding";
```

```
            break;
        case "ctr":
            mc = "DES/CTR/PKCS5Padding";
            md = "DES/CTR/NoPadding";
            break;
        default:
            break;
    }

    String sep = System.getProperty("file.separator");

    // GET TIME
    long TInicio, TFin, tiempo;
    // Take the begin
    TInicio = System.currentTimeMillis();

    /**** Cipher with mode CBC****/

    IvParameterSpec[] IV = new IvParameterSpec[3];

            /**First encryption**/
    // Generate key
    KeyGenerator keygenerator = KeyGenerator.getInstance("DES");
    keygenerator.init(56); // Size of key
    // Get key
    SecretKey desKey = keygenerator.generateKey();
    // Get base64 encoded version of the key
    String encodedKey =
     ↪  Base64.getEncoder().encodeToString(desKey.getEncoded());

    Cipher first_encryption = Cipher.getInstance(mc);
    first_encryption.init(Cipher.ENCRYPT_MODE, desKey);
    byte[] encryptedMessage_1 = first_encryption.doFinal(message);
    IV[0] = new IvParameterSpec(first_encryption.getIV());

            /**Decryption**/
    // Get key
    SecretKey desKey_2 = keygenerator.generateKey();
    // Get base64 encoded version of the key
    String encodedKey_2 =
     ↪  Base64.getEncoder().encodeToString(desKey_2.getEncoded());

    //Generate IV
    KeyGenerator ivGenerator = KeyGenerator.getInstance("DESede");
```

```
        ivGenerator.init(168);
        // Get IV
        SecretKey iv = keygenerator.generateKey();
        byte[] iv2 = iv.getEncoded();
        //Only use the first 8 bytes for the IV
        IV[1] = new IvParameterSpec(iv2, 0, 8);

        Cipher decryption = Cipher.getInstance(md);
        decryption.init(Cipher.DECRYPT_MODE, desKey_2, IV[1]);
        byte[] encryptedMessage_2 = decryption.doFinal(encryptedMessage_1);

                /**Second encryption**/
        Cipher second_encryption = Cipher.getInstance(mc);
        second_encryption.init(Cipher.ENCRYPT_MODE, desKey);
        byte[] encryptedMessage_final =
         ↪  first_encryption.doFinal(encryptedMessage_2);
        IV[2] = new IvParameterSpec(second_encryption.getIV());

        //File of the encrypted message
        String keys = encodedKey.concat(encodedKey_2);
        String cName = ".." + sep + "Encrypted" + sep + outFileName +
         ↪  "_2keys." + mode;
        String kFile_Name = ".." + sep + "Encrypted" + sep + outFileName +
         ↪  "_2keys_key." + mode;

        outFile(keys, IV, encryptedMessage_final, kFile_Name, cName);
        System.out.println("Succesfully encrypted in " + mode.toUpperCase()
         ↪  + "mode ");

        String[] files = {cName, kFile_Name};

        // TAKE TIME
        // Take end
        TFin = System.currentTimeMillis();
        tiempo = TFin - TInicio;
        System.out.println("Encryption time in ms: " + tiempo);

        return files;
    }
```

8.2 Decrypt

In the function where we decrypt with EDE with 2 keys we receive the following parameters:

- ✓ **String outFileName**: The name of the output file.
- ✓ **String cipherFile**: The name of the file with the message to decrypt.
- ✓ **String keyFile**: The name of the file that contains both keys.
- ✓ **String mode**: Indicate which mode of operation will be used.

As in the function to encrypt we first set the strings corresponding with the mode of operation especified, that will be passed to instances of Cipher to do the corresponding encryptions and decryptions.

After that we proceed to obtain the data from the files using a FileWriter for the file that contains the keys, and a FileInputStream (to obtain the stream of bytes) for the file that contains the IV's and the ciphertext.

As in the encryption function we create an array that stores the three IV's to pass as parameters to an instance of Cipher. Also we proceed to decode both keys.

Now, to decrypt, we do the inverse process. So, because in the encryption we Encrypt-Decrypt-Encrypt, now we Decrypt-Encrypt-Decrypt.

So for the first decryption we pass to the instance of Cipher the first key and the first IV. The next step is the encryption of the resultant message with the second key. Like in the encryption, in this step, using the second key, we add no padding, because of the problems that it could bring; also we pass to this instance of Cipher de second IV.

Finally we do the second decryption of the resultant message from the last step. This decryption is also done with the first key, and with the third IV. This last decryption will give us the final decrypted message.

The only thing left to do is create the output file with the decrypted message.

✓Code

```
public static void EDEdecipher_2Keys(String outFileName, String cipherFile,
 ↪  String keyFile, String mode) throws
            NoSuchAlgorithmException, NoSuchPaddingException,
             ↪  InvalidKeyException, InvalidAlgorithmParameterException,
            IllegalBlockSizeException, BadPaddingException, IOException{

        String mc = "", md = "";
        switch(mode){
            case "cbc":
                mc = "DES/CBC/PKCS5Padding";
                md = "DES/CBC/NoPadding";
                break;
            case "ofb":
                mc = "DES/OFB/PKCS5Padding";
```

```
            md = "DES/OFB/NoPadding";
            break;
        case "cfb":
            mc = "DES/CFB/PKCS5Padding";
            md = "DES/CFB/NoPadding";
            break;
        case "ctr":
            mc = "DES/CTR/PKCS5Padding";
            md = "DES/CTR/NoPadding";
            break;
        default:
            break;
    }

    String sep = System.getProperty("file.separator");

    // GET TIME
    long TInicio, TFin, tiempo;
    // Take the begin
    TInicio = System.currentTimeMillis();

    //Obtention of both keys from the file
    char[] key_1 = new char[12];
    char[] key_2 = new char[12];
    FileReader fk =new FileReader(keyFile);
    fk.read(key_1);
    fk.read(key_2);
    fk.close();

    byte[] message = null;
    byte[] iv_1 = new byte[8];
    byte[] iv_2 = new byte[8];
    byte[] iv_3 = new byte[8];
    File f =new File(cipherFile);
    try {
        FileInputStream in = new FileInputStream(f);
        in.read(iv_1, 0, 8);
        in.mark(8);
        in.read(iv_2, 0, 8);
        in.mark(16);
        in.read(iv_3, 0, 8);
        in.mark(24);
        message = new byte[(int)f.length() - 24];
        in.read(message, 0, message.length);
```

```
            in.close();
        }
        catch (Exception e) {
            e.printStackTrace();
        }

        IvParameterSpec[] IV = new IvParameterSpec[3];
        IV[0] = new IvParameterSpec(iv_1);
        IV[1] = new IvParameterSpec(iv_2);
        IV[2] = new IvParameterSpec(iv_3);

        // decode the base64 encoded strings
        byte[] decodedKey_1 =
         ↪  Base64.getDecoder().decode(String.copyValueOf(key_1));
        byte[] decodedKey_2 =
         ↪  Base64.getDecoder().decode(String.copyValueOf(key_2));
        // rebuild the keys using SecretKeySpec
        SecretKey originalKey_1 = new SecretKeySpec(decodedKey_1, 0,
         ↪  decodedKey_1.length, "DES");
        SecretKey originalKey_2 = new SecretKeySpec(decodedKey_2, 0,
         ↪  decodedKey_2.length, "DES");

        /****Decipher with mode CBC***/

                /**First decryption**/
        Cipher first_decryption = Cipher.getInstance(mc);
        first_decryption.init(Cipher.DECRYPT_MODE, originalKey_1, IV[0]);
        byte[] decryptedMessage_1 = first_decryption.doFinal(message);

                /**Encryption**/
        Cipher encryption = Cipher.getInstance(md);
        encryption.init(Cipher.ENCRYPT_MODE, originalKey_2, IV[1]);
        byte[] decryptedMessage_2 = encryption.doFinal(decryptedMessage_1);

                /**Second decryption**/
        Cipher second_decryption = Cipher.getInstance(mc);
        second_decryption.init(Cipher.DECRYPT_MODE, originalKey_1, IV[2]);
        byte[] decryptedMessage_final =
         ↪  first_decryption.doFinal(decryptedMessage_2);

        //File of the decrypted message
        String newName = ".." + sep+ "Decrypted" + sep + outFileName +
         ↪  "_2keys_d." + mode;
        if(createFile(decryptedMessage_final, newName))
```

```
            System.out.println("Succesfully decrypted in " +
             ↪  mode.toUpperCase() + " mode");
        else
            System.out.println("Error");

        // TAKE TIME
        // Take end
        TFin = System.currentTimeMillis();
        tiempo = TFin - TInicio;
        System.out.println("Decryption time in ms: " + tiempo);
    }
```

9 Functions used for EDE with 3 keys

9.1 Encrypt

We have a function to cipher with EDE. In this function we receive the following parameters:

- ✓ **byte[] message**: It contains the message to encrypt.
- ✓ **String outFileName**: The name of the output file.
- ✓ **String mode**: Indicate which mode of operation will be used.

For this case of the 3DES algorithm, we decided to use, instead of DES as the algorithm for the instance of Cipher, DESede. We didn't use it in the other cases because it is only available for the case Encryption-Decryption-Encryption, and only for a key lenght of 168 bits.

The advantage of use it is that we only need to generate a key, which will be passed as a parameter to the Cipher instance, and also we don't have to generate any IV, cause we only get it from the instance.

In esscence we do in this function the same as in the function for the encryption with 2 keys safe the changes mentioned, and also, because we only use a single instance of Cipher, instead of use predefined strings to set out the algorithm, mode of operation and padding scheme, we directly create the constructor with the mode of operation set as specified.

✓Code

```
public static String[] EDEcipher_3Keys(byte[] message, String outFileName,
 ↪  String mode) throws NoSuchAlgorithmException, NoSuchPaddingException,
            InvalidKeyException, InvalidAlgorithmParameterException,
             ↪  IllegalBlockSizeException, BadPaddingException, IOException{

        String sep = System.getProperty("file.separator");
```

```
// GET TIME
long TInicio, TFin, tiempo;
// Take the begin
TInicio = System.currentTimeMillis();

IvParameterSpec[] iv = new IvParameterSpec[1];

// Generate key
KeyGenerator keygenerator = KeyGenerator.getInstance("DESede");
keygenerator.init(168); // Size of key
// Get key
SecretKey desKey = keygenerator.generateKey();
// Get base64 encoded version of the key
String encodedKey =
 ↪  Base64.getEncoder().encodeToString(desKey.getEncoded());

// Cipher with mode CBC
Cipher desCipher = null;
switch(mode){
    case "cbc":
        desCipher = Cipher.getInstance("DESede/CBC/PKCS5Padding");
        break;
    case "ofb":
        desCipher = Cipher.getInstance("DESede/OFB/PKCS5Padding");
        break;
    case "cfb":
        desCipher = Cipher.getInstance("DESede/CFB/PKCS5Padding");
        break;
    case "ctr":
        desCipher = Cipher.getInstance("DESede/CTR/PKCS5Padding");
        break;
    default:
        break;
}
//Cipher desCipher = Cipher.getInstance("DESede/CBC/PKCS5Padding");
desCipher.init(Cipher.ENCRYPT_MODE, desKey);
byte[] encryptedMessage = desCipher.doFinal(message);

//File of the encrypted message
String cName = ".." + sep + "Encrypted" + sep + outFileName +
 ↪  "_3keys." + mode;
String kFile_Name = ".." + sep + "Encrypted" + sep + outFileName +
 ↪  "_3keys_key." + mode;
iv[0] = new IvParameterSpec(desCipher.getIV());
```

```
        outFile(encodedKey, iv, encryptedMessage, kFile_Name, cName);
        System.out.println("Succesfully encrypted in " + mode.toUpperCase()
        ↪  + "mode");

        String[] files = {cName, kFile_Name};

        // TAKE TIME
        // Take end
        TFin = System.currentTimeMillis();
        tiempo = TFin - TInicio;
        System.out.println("Encryption time in ms: " + tiempo);

        return files;
    }
```

9.2 Decrypt

In the function where we decrypt with EDE with 3 keys we receive the following parameters:

- ✓ **String outFileName**: The name of the output file.
- ✓ **String cipherFile**: The name of the file with the message to decrypt.
- ✓ **String keyFile**: The name of the file that contains both keys.
- ✓ **String mode**: Indicate which mode of operation will be used.

In this function we did in essence, the same as in the decryption function for EDE with 2 keys.

Some differences are:

- Instead of create an array that contains the IV's, because in this case we only use one IV, we passed directly the value recovered from the file as a parameter after store it in a IvParameterSpec.
- As in the encryption for the use of three keys, instead of using DES, as the algorithm specified for the instance of Cipher, we used DESede.
- Because of the use of the algorithm DESede, we didn't need to do Decrypt-Encrypt-Decrypt, just obtain the message generated for the instance Cipher using the DECRYPT MODE.

✓Code

```
public static void EDEdecipher_3Keys(String outFileName, String cipherFile,
 ↪  String keyFile, String mode) throws
            NoSuchAlgorithmException, NoSuchPaddingException,
             ↪  InvalidKeyException, InvalidAlgorithmParameterException,
            IllegalBlockSizeException, BadPaddingException, IOException{

        String sep = System.getProperty("file.separator");

        // GET TIME
        long TInicio, TFin, tiempo;
        // Take the begin
        TInicio = System.currentTimeMillis();

        //Obtention fo the key from the file
        char[] key = new char[32];
        FileReader fk =new FileReader(keyFile);
        fk.read(key);
        //Obtention of the IV and the ciphertext from the file
        byte[] message = null;
        byte[] iv = new byte[8];
        File f =new File(cipherFile);
        try {
            FileInputStream in = new FileInputStream(f);
            in.read(iv, 0, 8);
            message = new byte[(int)f.length() - 8];
            in.mark(8);
            in.read(message, 0, message.length);
            in.close();
        }
        catch (Exception e) {
            e.printStackTrace();
        }

        IvParameterSpec IVparam = new IvParameterSpec(iv);

        // decode the base64 encoded string
        byte[] decodedKey =
         ↪  Base64.getDecoder().decode(String.copyValueOf(key));
        // rebuild key using SecretKeySpec
        SecretKey originalKey = new SecretKeySpec(decodedKey, 0,
         ↪  decodedKey.length, "DESede");

        // Decipher with mode CBC
        Cipher desCipher = null;
```

```
        switch(mode){
            case "cbc":
                desCipher = Cipher.getInstance("DESede/CBC/PKCS5Padding");
                break;
            case "ofb":
                desCipher = Cipher.getInstance("DESede/OFB/PKCS5Padding");
                break;
            case "cfb":
                desCipher = Cipher.getInstance("DESede/CFB/PKCS5Padding");
                break;
            case "ctr":
                desCipher = Cipher.getInstance("DESede/CTR/PKCS5Padding");
                break;
            default:
                break;
        }
        desCipher.init(Cipher.DECRYPT_MODE, originalKey, IVparam);
        byte[] decryptedMessage = desCipher.doFinal(message);

        //File of the decrypted message
        String newName = ".." + sep+ "Decrypted" + sep + outFileName +
         ↪  "_3keys_d." + mode;
        if(createFile(decryptedMessage, newName))
            System.out.println("Succesfully decrypted in " +
             ↪  mode.toUpperCase() + " mode");
        else
            System.out.println("Error");

        // TAKE TIME
        // Take end
        TFin = System.currentTimeMillis();
        tiempo = TFin - TInicio;
        System.out.println("Decryption time in ms: " + tiempo);
    }
```

10 Testing

10.1 EEE with 2 keys

10.2 CBC

```
-----> Working with:sistemas_operativos.pdf

File: sistemas_operativos.pdf Size: 12042469 bytes.
Time in MS: 4470


-----> Working with:Mov_Pendulo_Simple.mov

File: Mov_Pendulo_Simple.mov Size: 10472710 bytes.
Time in MS: 2363


-----> Working with:ArquitecturaComputadoras.PDF

File: ArquitecturaComputadoras.PDF Size: 5135588 bytes.
Time in MS: 1105


-----> Working with:Recursos_Apuntes_21_Vectorial.doc

File: Recursos_Apuntes_21_Vectorial.doc Size: 3309056 bytes.
Time in MS: 828


-----> Working with:Aparato_Respiratorio.pptm

File: Aparato_Respiratorio.pptm Size: 1572582 bytes.
Time in MS: 437


-----> Working with:ha.exe

File: ha.exe Size: 1053966 bytes.
Time in MS: 343


-----> Working with:guia-tkinter.pdf

File: guia-tkinter.pdf Size: 517168 bytes.
Time in MS: 157


-----> Working with:ipn.PNG

File: ipn.PNG Size: 426013 bytes.
Time in MS: 156
```

10.3 CTR

```
-----> Working with:sistemas_operativos.pdf

File: sistemas_operativos.pdf Size: 12042469 bytes.
Time in MS: 4658


-----> Working with:Mov_Pendulo_Simple.mov

File: Mov_Pendulo_Simple.mov Size: 10472710 bytes.
Time in MS: 2468


-----> Working with:ArquitecturaComputadoras.PDF

File: ArquitecturaComputadoras.PDF Size: 5135588 bytes.
Time in MS: 1289


-----> Working with:Recursos_Apuntes_21_Vectorial.doc

File: Recursos_Apuntes_21_Vectorial.doc Size: 3309056 bytes.
Time in MS: 758


-----> Working with:Aparato_Respiratorio.pptm

File: Aparato_Respiratorio.pptm Size: 1572582 bytes.
Time in MS: 406


-----> Working with:ha.exe

File: ha.exe Size: 1053966 bytes.
Time in MS: 266


-----> Working with:guia-tkinter.pdf

File: guia-tkinter.pdf Size: 517168 bytes.
Time in MS: 125


-----> Working with:ipn.PNG

File: ipn.PNG Size: 426013 bytes.
Time in MS: 109
```

10.4 OFB

```
-----> Working with:sistemas_operativos.pdf

File: sistemas_operativos.pdf Size: 12042469 bytes.
Time in MS: 4495


-----> Working with:Mov_Pendulo_Simple.mov

File: Mov_Pendulo_Simple.mov Size: 10472710 bytes.
Time in MS: 2391


-----> Working with:ArquitecturaComputadoras.PDF

File: ArquitecturaComputadoras.PDF Size: 5135588 bytes.
Time in MS: 1296


-----> Working with:Recursos_Apuntes_21_Vectorial.doc

File: Recursos_Apuntes_21_Vectorial.doc Size: 3309056 bytes.
Time in MS: 719


-----> Working with:Aparato_Respiratorio.pptm

File: Aparato_Respiratorio.pptm Size: 1572582 bytes.
Time in MS: 369


-----> Working with:ha.exe

File: ha.exe Size: 1053966 bytes.
Time in MS: 234


-----> Working with:guia-tkinter.pdf

File: guia-tkinter.pdf Size: 517168 bytes.
Time in MS: 147


-----> Working with:ipn.PNG

File: ipn.PNG Size: 426013 bytes.
Time in MS: 93
```

10.5 CFB

```
-----> Working with:sistemas_operativos.pdf

File: sistemas_operativos.pdf Size: 12042469 bytes.
Time in MS: 4578


-----> Working with:Mov_Pendulo_Simple.mov

File: Mov_Pendulo_Simple.mov Size: 10472710 bytes.
Time in MS: 2391


-----> Working with:ArquitecturaComputadoras.PDF

File: ArquitecturaComputadoras.PDF Size: 5135588 bytes.
Time in MS: 1109


-----> Working with:Recursos_Apuntes_21_Vectorial.doc

File: Recursos_Apuntes_21_Vectorial.doc Size: 3309056 bytes.
Time in MS: 719


-----> Working with:Aparato_Respiratorio.pptm

File: Aparato_Respiratorio.pptm Size: 1572582 bytes.
Time in MS: 374


-----> Working with:ha.exe

File: ha.exe Size: 1053966 bytes.
Time in MS: 250


-----> Working with:guia-tkinter.pdf

File: guia-tkinter.pdf Size: 517168 bytes.
Time in MS: 141


-----> Working with:ipn.PNG

File: ipn.PNG Size: 426013 bytes.
Time in MS: 93
```

10.6 EEE with 3 keys

10.7 CBC

```
-----> Working with:sistemas_operativos.pdf

File: sistemas_operativos.pdf Size: 12042469 bytes.
Time in MS: 4780


-----> Working with:Mov_Pendulo_Simple.mov

File: Mov_Pendulo_Simple.mov Size: 10472710 bytes.
Time in MS: 2413


-----> Working with:ArquitecturaComputadoras.PDF

File: ArquitecturaComputadoras.PDF Size: 5135588 bytes.
Time in MS: 1187


-----> Working with:Recursos_Apuntes_21_Vectorial.doc

File: Recursos_Apuntes_21_Vectorial.doc Size: 3309056 bytes.
Time in MS: 727


-----> Working with:Aparato_Respiratorio.pptm

File: Aparato_Respiratorio.pptm Size: 1572582 bytes.
Time in MS: 375


-----> Working with:ha.exe

File: ha.exe Size: 1053966 bytes.
Time in MS: 282


-----> Working with:guia-tkinter.pdf

File: guia-tkinter.pdf Size: 517168 bytes.
Time in MS: 328


-----> Working with:ipn.PNG

File: ipn.PNG Size: 426013 bytes.
Time in MS: 204
```

10.8 CTR

```
-----> Working with:sistemas_operativos.pdf

File: sistemas_operativos.pdf Size: 12042469 bytes.
Time in MS: 5073


-----> Working with:Mov_Pendulo_Simple.mov

File: Mov_Pendulo_Simple.mov Size: 10472710 bytes.
Time in MS: 2599


-----> Working with:ArquitecturaComputadoras.PDF

File: ArquitecturaComputadoras.PDF Size: 5135588 bytes.
Time in MS: 1417


-----> Working with:Recursos_Apuntes_21_Vectorial.doc

File: Recursos_Apuntes_21_Vectorial.doc Size: 3309056 bytes.
Time in MS: 1358


-----> Working with:Aparato_Respiratorio.pptm

File: Aparato_Respiratorio.pptm Size: 1572582 bytes.
Time in MS: 555


-----> Working with:ha.exe

File: ha.exe Size: 1053966 bytes.
Time in MS: 375


-----> Working with:guia-tkinter.pdf

File: guia-tkinter.pdf Size: 517168 bytes.
Time in MS: 157


-----> Working with:ipn.PNG

File: ipn.PNG Size: 426013 bytes.
Time in MS: 109
```

10.9 OFB

```
-----> Working with:sistemas_operativos.pdf

File: sistemas_operativos.pdf Size: 12042469 bytes.
Time in MS: 4885


-----> Working with:Mov_Pendulo_Simple.mov

File: Mov_Pendulo_Simple.mov Size: 10472710 bytes.
Time in MS: 2448


-----> Working with:ArquitecturaComputadoras.PDF

File: ArquitecturaComputadoras.PDF Size: 5135588 bytes.
Time in MS: 1192


-----> Working with:Recursos_Apuntes_21_Vectorial.doc

File: Recursos_Apuntes_21_Vectorial.doc Size: 3309056 bytes.
Time in MS: 938


-----> Working with:Aparato_Respiratorio.pptm

File: Aparato_Respiratorio.pptm Size: 1572582 bytes.
Time in MS: 359


-----> Working with:ha.exe

File: ha.exe Size: 1053966 bytes.
Time in MS: 314


-----> Working with:guia-tkinter.pdf

File: guia-tkinter.pdf Size: 517168 bytes.
Time in MS: 141


-----> Working with:ipn.PNG

File: ipn.PNG Size: 426013 bytes.
Time in MS: 109
```

10.10 CFB

```
-----> Working with:sistemas_operativos.pdf

File: sistemas_operativos.pdf Size: 12042469 bytes.
Time in MS: 5120


-----> Working with:Mov_Pendulo_Simple.mov

File: Mov_Pendulo_Simple.mov Size: 10472710 bytes.
Time in MS: 2420


-----> Working with:ArquitecturaComputadoras.PDF

File: ArquitecturaComputadoras.PDF Size: 5135588 bytes.
Time in MS: 1237


-----> Working with:Recursos_Apuntes_21_Vectorial.doc

File: Recursos_Apuntes_21_Vectorial.doc Size: 3309056 bytes.
Time in MS: 890


-----> Working with:Aparato_Respiratorio.pptm

File: Aparato_Respiratorio.pptm Size: 1572582 bytes.
Time in MS: 625


-----> Working with:ha.exe

File: ha.exe Size: 1053966 bytes.
Time in MS: 360


-----> Working with:guia-tkinter.pdf

File: guia-tkinter.pdf Size: 517168 bytes.
Time in MS: 218


-----> Working with:ipn.PNG

File: ipn.PNG Size: 426013 bytes.
Time in MS: 141
```

10.11 EDE with 2 keys

10.12 CBC

```
Name of the file to encrypt: sistemas_operativos.pdf
El archivo tiene 12042469 de bytes.
Encryption time in ms: 1830
Decryption time in ms: 1137

Name of the file to encrypt: Mov_Pendulo_Simple.mov
El archivo tiene 10472710 de bytes.
Encryption time in ms: 876
Decryption time in ms: 866

Name of the file to encrypt: ArquitecturaComputadoras.pdf
El archivo tiene 5135588 de bytes.
Encryption time in ms: 551
Decryption time in ms: 553

Name of the file to encrypt: Recursos_Apuntes_21_Vectorial.doc
El archivo tiene 3309056 de bytes.
Encryption time in ms: 376
Decryption time in ms: 406

Name of the file to encrypt: Aparato_Respiratorio.pptm
El archivo tiene 1572582 de bytes.
Encryption time in ms: 211
Decryption time in ms: 170

Name of the file to encrypt: ha.exe
El archivo tiene 1053966 de bytes.
Encryption time in ms: 93
Decryption time in ms: 89

Name of the file to encrypt: guia-tkinter.pdf
El archivo tiene 517168 de bytes.
Encryption time in ms: 101
Decryption time in ms: 121

Name of the file to encrypt: ipn.png
El archivo tiene 426013 de bytes.
Encryption time in ms: 90
Decryption time in ms: 77
```

10.13 CTR

```
Name of the file to encrypt: sistemas_operativos.pdf
El archivo tiene 12042469 de bytes.
Encryption time in ms: 1055
Decryption time in ms: 1199

Name of the file to encrypt: Mov_Pendulo_Simple.mov
El archivo tiene 10472710 de bytes.
Encryption time in ms: 1082
Decryption time in ms: 910

Name of the file to encrypt: ArquitecturaComputadoras.pdf
El archivo tiene 5135588 de bytes.
Encryption time in ms: 479
Decryption time in ms: 484

Name of the file to encrypt: Recursos_Apuntes_21_Vectorial.doc
El archivo tiene 3309056 de bytes.
Encryption time in ms: 393
Decryption time in ms: 438

Name of the file to encrypt: Aparato_Respiratorio.pptm
El archivo tiene 1572582 de bytes.
Encryption time in ms: 221
Decryption time in ms: 165

Name of the file to encrypt: ha.exe
El archivo tiene 1053966 de bytes.
Encryption time in ms: 104
Decryption time in ms: 98

Name of the file to encrypt: guia-tkinter.pdf
El archivo tiene 517168 de bytes.
Encryption time in ms: 118
Decryption time in ms: 73

Name of the file to encrypt: ipn.png
El archivo tiene 426013 de bytes.
Encryption time in ms: 67
Decryption time in ms: 58
```

10.14 OFB

```
Name of the file to encrypt: sistemas_operativos.pdf
El archivo tiene 12042469 de bytes.
Encryption time in ms: 1002
Decryption time in ms: 1074

Name of the file to encrypt: Mov_Pendulo_Simple.mov
El archivo tiene 10472710 de bytes.
Encryption time in ms: 919
Decryption time in ms: 904

Name of the file to encrypt: ArquitecturaComputadoras.pdf
El archivo tiene 5135588 de bytes.
Encryption time in ms: 451
Decryption time in ms: 496

Name of the file to encrypt: Recursos_Apuntes_21_Vectorial.doc
El archivo tiene 3309056 de bytes.
Encryption time in ms: 299
Decryption time in ms: 336

Name of the file to encrypt: Aparato_Respiratorio.pptm
El archivo tiene 1572582 de bytes.
Encryption time in ms: 147
Decryption time in ms: 182

Name of the file to encrypt: ha.exe
El archivo tiene 1053966 de bytes.
Encryption time in ms: 83
Decryption time in ms: 107

Name of the file to encrypt: guia-tkinter.pdf
El archivo tiene 517168 de bytes.
Encryption time in ms: 81
Decryption time in ms: 79

Name of the file to encrypt: ipn.png
El archivo tiene 426013 de bytes.
Encryption time in ms: 58
Decryption time in ms: 48
```

10.15 CFB

```
Name of the file to encrypt: sistemas_operativos.pdf
El archivo tiene 12042469 de bytes.
Encryption time in ms: 984
Decryption time in ms: 1086

Name of the file to encrypt: Mov_Pendulo_Simple.mov
El archivo tiene 10472710 de bytes.
Encryption time in ms: 1156
Decryption time in ms: 914

Name of the file to encrypt: ArquitecturaComputadoras.pdf
El archivo tiene 5135588 de bytes.
Encryption time in ms: 536
Decryption time in ms: 434

Name of the file to encrypt: Recursos_Apuntes_21_Vectorial.doc
El archivo tiene 3309056 de bytes.
Encryption time in ms: 338
Decryption time in ms: 306

Name of the file to encrypt: Aparato_Respiratorio.pptm
El archivo tiene 1572582 de bytes.
Encryption time in ms: 161
Decryption time in ms: 125

Name of the file to encrypt: ha.exe
El archivo tiene 1053966 de bytes.
Encryption time in ms: 153
Decryption time in ms: 113

Name of the file to encrypt: guia-tkinter.pdf
El archivo tiene 517168 de bytes.
Encryption time in ms: 75
Decryption time in ms: 78

Name of the file to encrypt: ipn.png
El archivo tiene 426013 de bytes.
Encryption time in ms: 37
Decryption time in ms: 39
```

10.16 EDE with 3 keys

10.17 CBC

```
Name of the file to encrypt: sistemas_operativos.pdf
El archivo tiene 12042469 de bytes.
Encryption time in ms: 1777
Decryption time in ms: 1138

Name of the file to encrypt: Mov_Pendulo_Simple.mov
El archivo tiene 10472710 de bytes.
Encryption time in ms: 825
Decryption time in ms: 787

Name of the file to encrypt: ArquitecturaComputadoras.pdf
El archivo tiene 5135588 de bytes.
Encryption time in ms: 429
Decryption time in ms: 397

Name of the file to encrypt: Recursos_Apuntes_21_Vectorial.doc
El archivo tiene 3309056 de bytes.
Encryption time in ms: 329
Decryption time in ms: 280

Name of the file to encrypt: Aparato_Respiratorio.pptm
El archivo tiene 1572582 de bytes.
Encryption time in ms: 180
Decryption time in ms: 150

Name of the file to encrypt: ha.exe
El archivo tiene 1053966 de bytes.
Encryption time in ms: 170
Decryption time in ms: 115

Name of the file to encrypt: guia-tkinter.pdf
El archivo tiene 517168 de bytes.
Encryption time in ms: 50
Decryption time in ms: 91

Name of the file to encrypt: ipn.png
El archivo tiene 426013 de bytes.
Encryption time in ms: 43
Decryption time in ms: 40
```

10.18 CTR

```
Name of the file to encrypt: sistemas_operativos.pdf
El archivo tiene 12042469 de bytes.
Encryption time in ms: 1223
Decryption time in ms: 1202

Name of the file to encrypt: Mov_Pendulo_Simple.mov
El archivo tiene 10472710 de bytes.
Encryption time in ms: 919
Decryption time in ms: 924

Name of the file to encrypt: ArquitecturaComputadoras.pdf
El archivo tiene 5135588 de bytes.
Encryption time in ms: 459
Decryption time in ms: 541

Name of the file to encrypt: Recursos_Apuntes_21_Vectorial.doc
El archivo tiene 3309056 de bytes.
Encryption time in ms: 304
Decryption time in ms: 282

Name of the file to encrypt: Aparato_Respiratorio.pptm
El archivo tiene 1572582 de bytes.
Encryption time in ms: 160
Decryption time in ms: 130

Name of the file to encrypt: ha.exe
El archivo tiene 1053966 de bytes.
Encryption time in ms: 85
Decryption time in ms: 105

Name of the file to encrypt: guia-tkinter.pdf
El archivo tiene 517168 de bytes.
Encryption time in ms: 83
Decryption time in ms: 50

Name of the file to encrypt: ipn.png
El archivo tiene 426013 de bytes.
Encryption time in ms: 102
Decryption time in ms: 60
```

10.19 OFB

```
Name of the file to encrypt: sistemas_operativos.pdf
El archivo tiene 12042469 de bytes.
Encryption time in ms: 1161
Decryption time in ms: 1197

Name of the file to encrypt: Mov_Pendulo_Simple.mov
El archivo tiene 10472710 de bytes.
Encryption time in ms: 872
Decryption time in ms: 868

Name of the file to encrypt: ArquitecturaComputadoras.pdf
El archivo tiene 5135588 de bytes.
Encryption time in ms: 473
Decryption time in ms: 448

Name of the file to encrypt: Recursos_Apuntes_21_Vectorial.doc
El archivo tiene 3309056 de bytes.
Encryption time in ms: 320
Decryption time in ms: 319

Name of the file to encrypt: Aparato_Respiratorio.pptm
El archivo tiene 1572582 de bytes.
Encryption time in ms: 159
Decryption time in ms: 130

Name of the file to encrypt: ha.exe
El archivo tiene 1053966 de bytes.
Encryption time in ms: 90
Decryption time in ms: 100

Name of the file to encrypt: guia-tkinter.pdf
El archivo tiene 517168 de bytes.
Encryption time in ms: 40
Decryption time in ms: 40

Name of the file to encrypt: ipn.png
El archivo tiene 426013 de bytes.
Encryption time in ms: 40
Decryption time in ms: 40
```

10.20 CFB

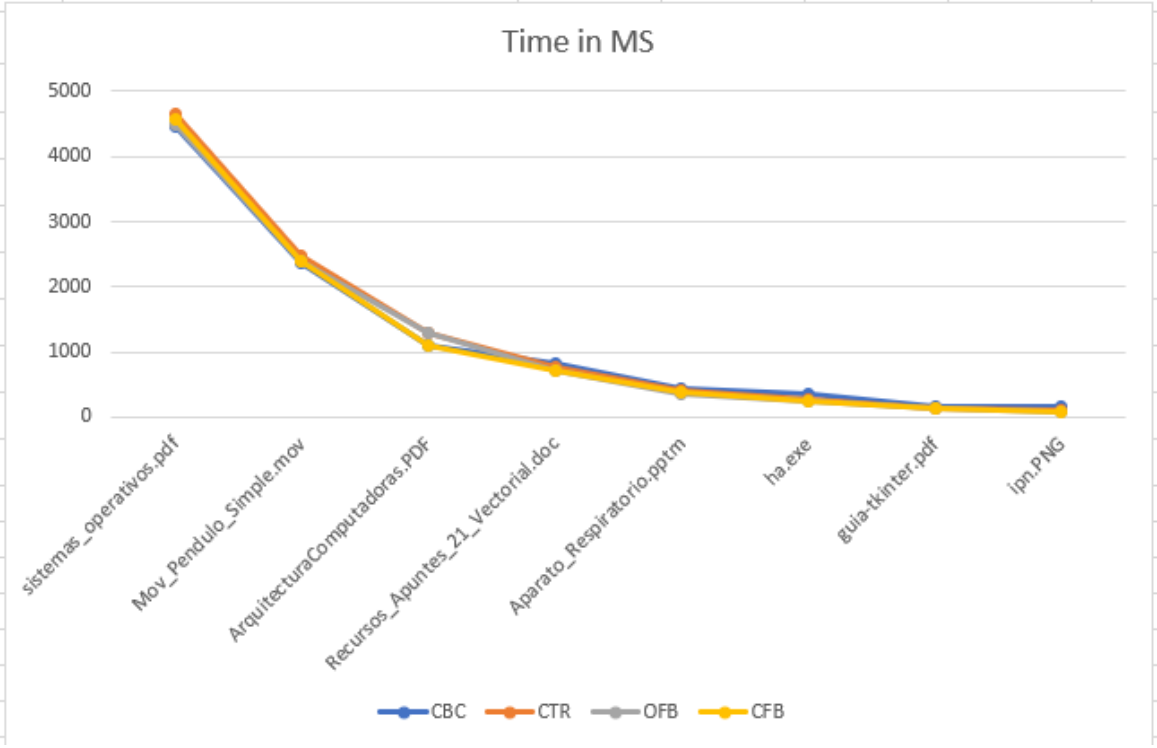
```
Name of the file to encrypt: sistemas_operativos.pdf
El archivo tiene 12042469 de bytes.
Encryption time in ms: 1133
Decryption time in ms: 1183

Name of the file to encrypt: Mov_Pendulo_Simple.mov
El archivo tiene 10472710 de bytes.
Encryption time in ms: 802
Decryption time in ms: 935

Name of the file to encrypt: ArquitecturaComputadoras.pdf
El archivo tiene 5135588 de bytes.
Encryption time in ms: 519
Decryption time in ms: 410

Name of the file to encrypt: Recursos_Apuntes_21_Vectorial.doc
El archivo tiene 3309056 de bytes.
Encryption time in ms: 287
Decryption time in ms: 293

Name of the file to encrypt: Aparato_Respiratorio.pptm
El archivo tiene 1572582 de bytes.
Encryption time in ms: 150
Decryption time in ms: 150

Name of the file to encrypt: ha.exe
El archivo tiene 1053966 de bytes.
Encryption time in ms: 90
Decryption time in ms: 125

Name of the file to encrypt: guia-tkinter.pdf
El archivo tiene 517168 de bytes.
Encryption time in ms: 50
Decryption time in ms: 40

Name of the file to encrypt: ipn.png
El archivo tiene 426013 de bytes.
Encryption time in ms: 35
Decryption time in ms: 40
```

11 Graphs

11.1 EEE

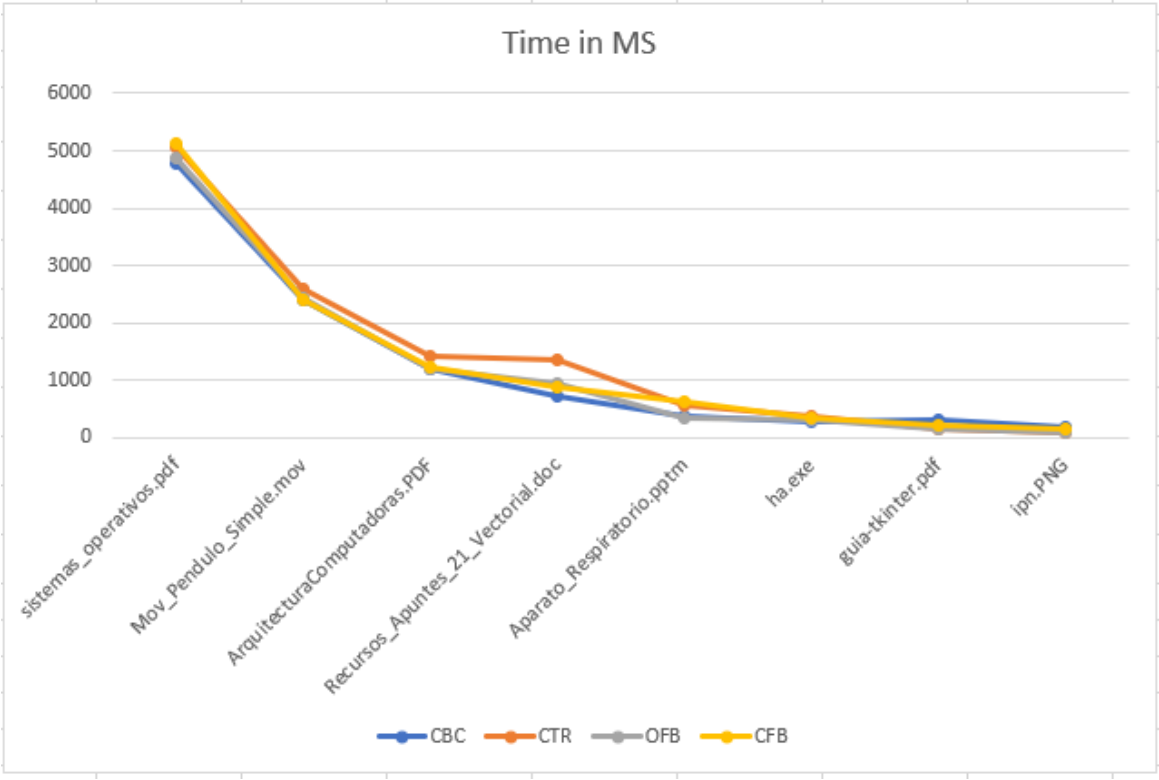
11.2 With 2 keys

3DES EEE with 2 keys				
Mode Operation/ File	**CBC**	**CTR**	**OFB**	**CFB**
sistemas_operativos.pdf	4470	4658	4495	4578
Mov_Pendulo_Simple.mov	2363	2468	2391	2391
ArquitecturaComputadoras.PDF	1105	1289	1296	1109
Recursos_Apuntes_21_Vectorial.doc	828	758	719	719
Aparato_Respiratorio.pptm	437	406	369	374
ha.exe	343	266	234	250
guia-tkinter.pdf	157	125	147	141
ipn.PNG	156	109	93	93



11.3 With 3 keys

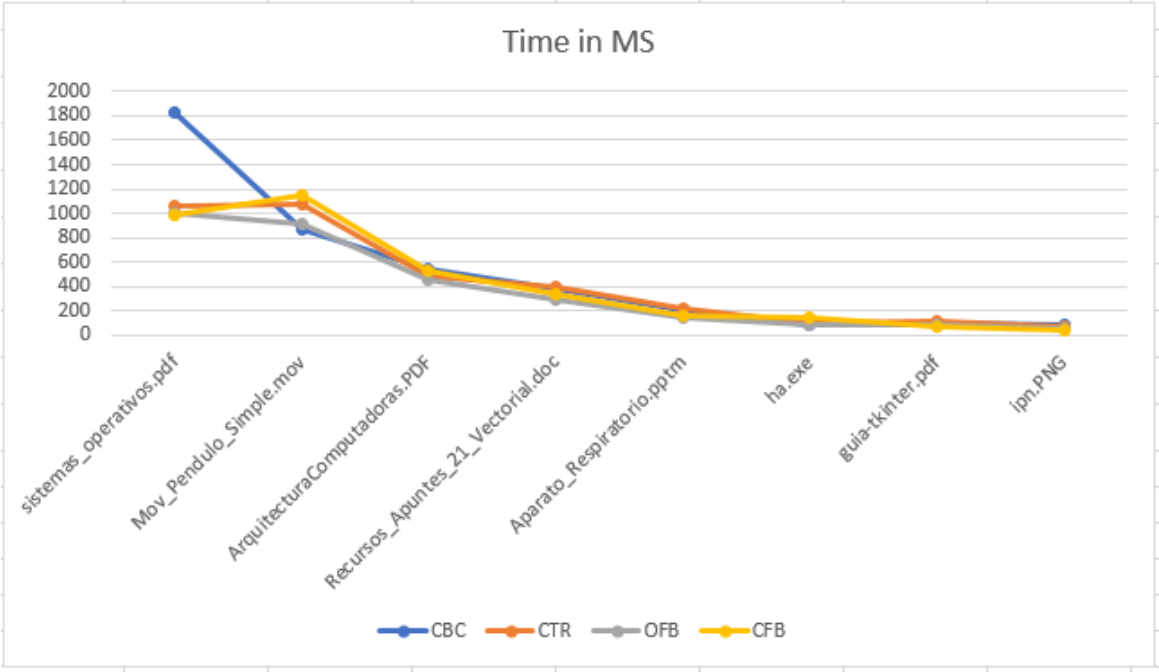
3DES EEE with 3 keys				
Mode Operation/ File	**CBC**	**CTR**	**OFB**	**CFB**
sistemas_operativos.pdf	4780	5073	4885	5120
Mov_Pendulo_Simple.mov	2413	2599	2448	2420
ArquitecturaComputadoras.PDF	1187	1417	1192	1237
Recursos_Apuntes_21_Vectorial.doc	727	1358	938	890
Aparato_Respiratorio.pptm	375	555	359	625
ha.exe	282	375	314	360
guia-tkinter.pdf	328	157	141	218
ipn.PNG	204	109	109	141



11.4 EDE

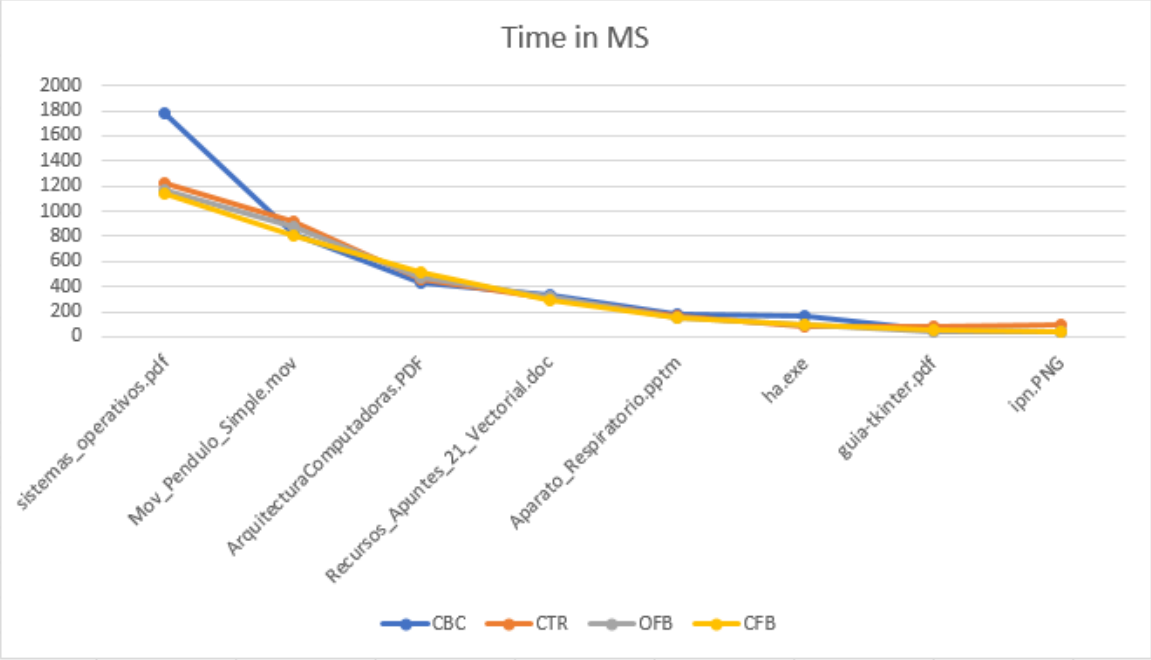
11.5 With 2 keys

3DES EDE with 2 keys				
Mode Operation/ File	CBC	CTR	OFB	CFB
sistemas_operativos.pdf	1830	1055	1002	984
Mov_Pendulo_Simple.mov	876	1082	919	1156
ArquitecturaComputadoras.PDF	551	479	451	536
Recursos_Apuntes_21_Vectorial.doc	376	393	299	338
Aparato_Respiratorio.pptm	211	221	147	161
ha.exe	93	104	83	153
guia-tkinter.pdf	101	118	81	75
ipn.PNG	90	67	58	37



11.6 With 3 keys

3DES EDE with 3 keys				
Mode Operation/ File	CBC	CTR	OFB	CFB
sistemas_operativos.pdf	1777	1223	1161	1133
Mov_Pendulo_Simple.mov	825	919	872	802
ArquitecturaComputadoras.PDF	429	459	473	519
Recursos_Apuntes_21_Vectorial.doc	329	304	320	287
Aparato_Respiratorio.pptm	180	160	159	150
ha.exe	170	85	90	90
guia-tkinter.pdf	50	83	40	50
ipn.PNG	43	102	40	35



12 Conclusion

Speaking generally, comparing the times showed for the EEE variant and the EDE variant:

- We saw that independently of the mode of operation used in whatever variation of the 3DES algorithm, every mode will show a minimum variation for the others in the time that took the encryption. But all of them will took approximately the same time.

- Other thing we noticed is that our cipher in the mode EEE took more time for the encryption that our cipher in the mode EDE, for both 2 and 3 keys.

Now talking about specifically of the EEE variation we noticed that for the variant with 3 keys, it took more time to encrypt that with 2 keys. This is because of the generation of the extra key and the extra IV.

About the EDE variation we didn't notice so much change in the times for the encryption. Sometimes it took more time for the variation with 2 keys, and other times for the variation with 3 keys. We suppose that this "balance" could be because of the use of the already implemented algorithm DESede, that there are more steps that this algorithm does, in comparation with the DES algorithm, but we coded less operations that for our implementation of EDE with 2 keys.

References

[1] D. Chakraborty and F. R. Henriquez, *Block Cipher Modes of Operation from Hardware Implementation Perspective*.