

TUTORIAL

CRYPTOGRAPH

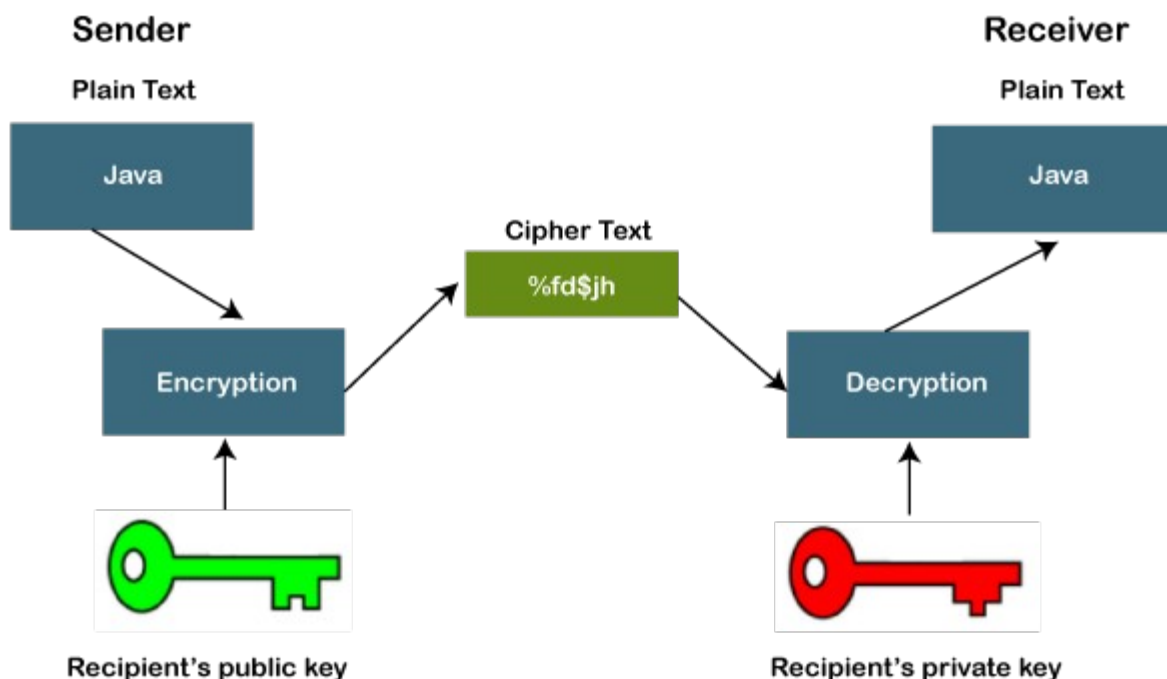
Y

Symmetric Key Cryptography

I. Java Code for DES

Our dependency on the internet is increasing day by day and we share lots of personal information with others. Since our data or personal information is not secure. For this reason, the security of the data become essential for us. We need to keep data confidential, unmodified, and readily available to authorized readers only. We can make secure data by using the **DES (Data Encryption Standard)** mechanism that can **encrypt** and **decrypt** the data. Using the **DES algorithm** is the most popular way to encrypt and decrypt data. It is a widely used **symmetric** (encryption and decryption) algorithm in the world.

In this section, we will learn the **DES algorithm** that is used to generate the **ciphertext**. Also, we will implement the **DES algorithm** in a **Java** program.



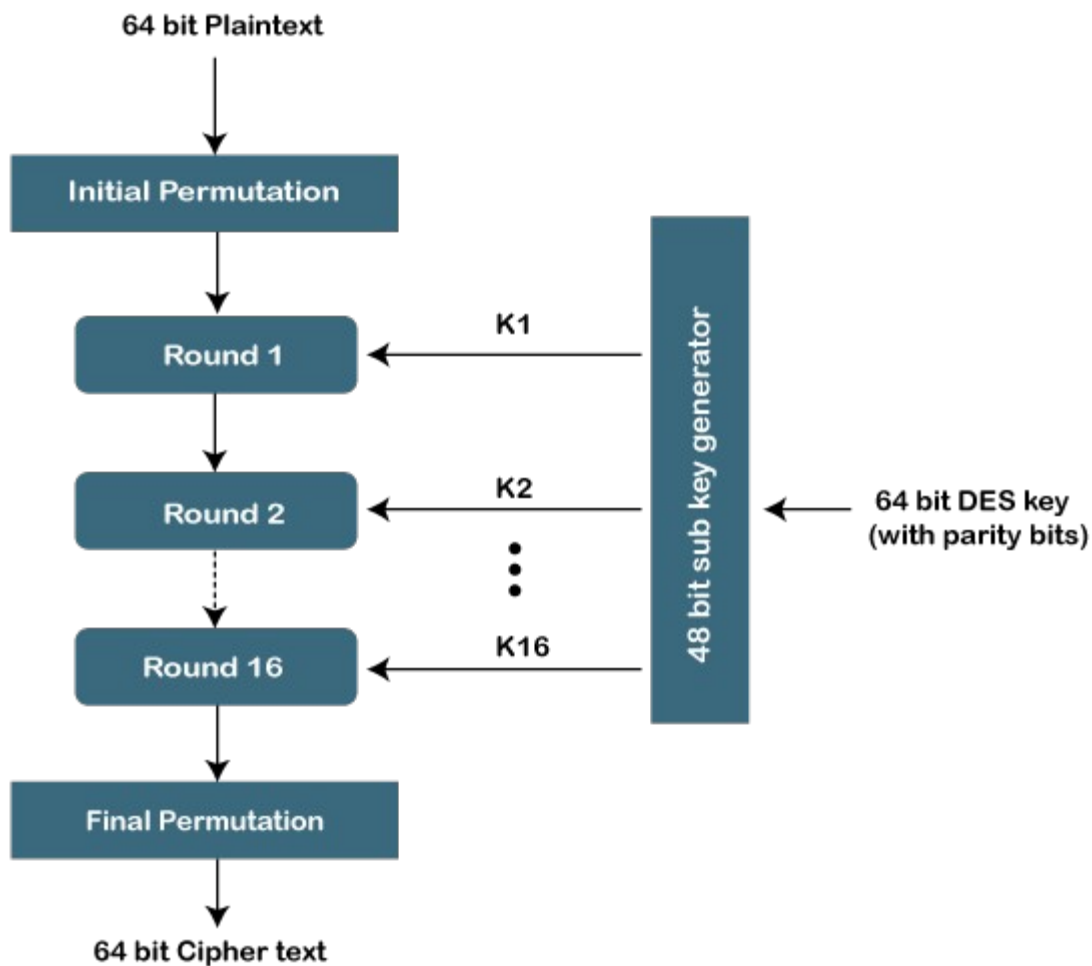
DES Algorithm

DES stands for Data Encryption Standard. It is a symmetric-key block cipher algorithm used to encrypt and decrypt data. It is developed by the IBM team in early 1970. It accepts the plaintext in 64-bit blocks and changes it into the ciphertext that uses the 64-bit keys to encrypt the data. The algorithm uses the same key to encrypt and decrypt the data.

It is based on **LUCIFER** (also known as Feistel block cipher algorithm) which is a direct predecessor of the DES algorithm. It is developed by eminent scholar and researcher **Horst Feistel** at IBM. It provides high security by using a 128-bit key block and a 128-bit block size. The DES algorithm uses the 16 rounds of the **Feistel structure**. The structure uses a unique key for each round. Finally, in 1976, it was approved by the federal encryption standard.

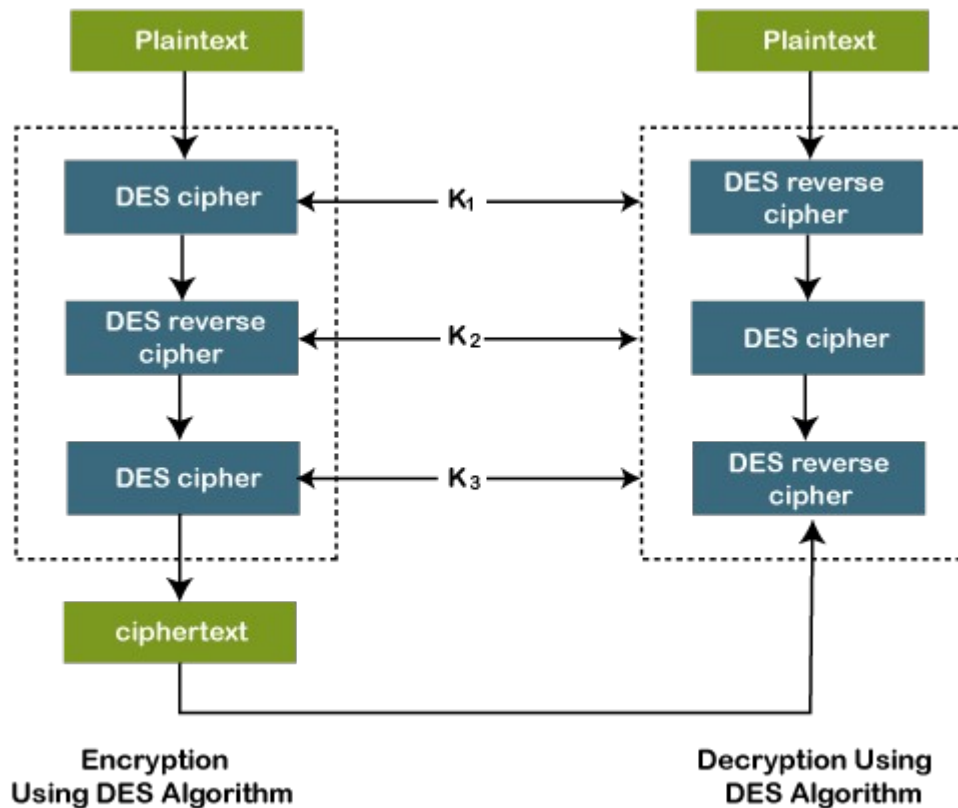
In 2002, AES (Advanced Encryption Standard) replaced the DES encryption algorithm as the accepted standard. Later in 1995, the advanced version of the DES algorithm was introduced that is known as **Triple DES** (3DES or TDES). Officially, it is known as **Triple Data Encryption Algorithm** (TDEA or 3DEA).

TDEA is also a symmetric-key block cipher algorithm that uses the DES cipher algorithm thrice to each data block. Its block size is **64-bits** and key sizes are 168, 112, and 56-bits, respectively for the keys 1, 2, and 3. It also uses the DES equivalent rounds i.e. 48. It means 16 rounds for each key.



It encrypts the data using the first key (k1), decrypts the data by using the second key (k2) and again encrypts the data by using the third key (k3). Another variant of the algorithm uses only two keys k1 and k3. Where both the keys k1 and k3 are the same. It is used still but considered as a legacy algorithm.

The following figure shows the encryption and decryption using TDEA.



Let's understand the DES algorithm.

Generating Keys

The algorithm performs 16 rounds of encryption and for each round, a unique key is generated. Before moving to the steps, it is important to know that in plaintext the bits are labeled from 1 to 64 where 1 is the most significant bit and 64 is the least significant bit. The process of generating keys are as follows:

1. First, we compress and transpose the given 64-bit key into a 48-bit keys by using the table given below:

```

int pc1[56] = {
    57,49,41,33,25,17,9,
    1,58,50,42,34,26,18,
    10,2,59,51,43,35,27,
    19,11,3,60,52,44,36,
    63,55,47,39,31,23,15,
    7,62,54,46,38,30,22,
    14,6,61,53,45,37,29,
    21,13,5,28,20,12,4
  }
  
```

};

2. Separate the result into two equal parts i.e. C and D.

3. The part C and D are left-shifted circularly. For encryption, the 1st, 2nd, 9th, and 16th round is responsible that shifts a bit to the left by 1 bit, circularly. All the rest rounds are shifted to the left by 2-bit circularly.

4. After that, the result is compressed to 48-bits with the help of the following table.

1. `int pc2[48] = {`
2. `14,17,11,24,1,5,`
3. `3,28,15,6,21,10,`
4. `23,19,12,4,26,8,`
5. `16,7,27,20,13,2,`
6. `41,52,31,37,47,55,`
7. `30,40,51,45,33,48,`
8. `44,49,39,56,34,53,`
9. `46,42,50,36,29,32`
10. `};`

5. The result that we get from step 3 becomes the input for the next round of the key generation.

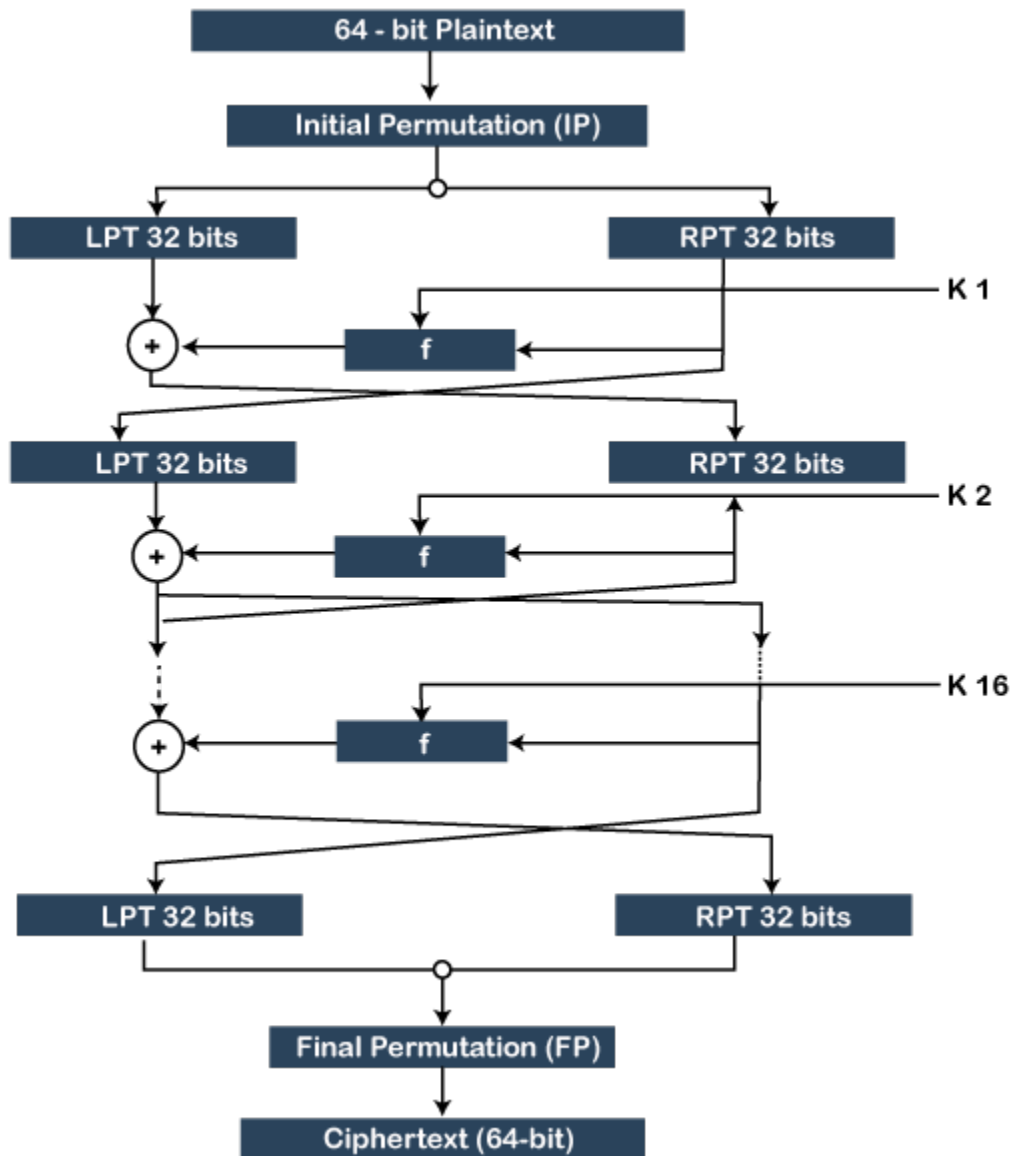
Encryption Steps of the Algorithm

The algorithm includes the following steps:

1. The algorithm takes the 64-bit plain text as input.
2. The text is parsed into a function called the Initial Permutation (IP) function.
3. The initial permutation (IP) function breaks the plain text into the two halves of the permuted block. These two blocks are known as Left Plain Text (LPT) and Right Plain Text (RPT).
4. The 16 round encryption process is performed on both blocks LPT and RPT. The encryption process performs the following:
 1. Key Transformation
 2. Expansion Permutation
 3. S-Box Permutation
 4. P-Box Permutation

5. XOR and Swap

- After performing the encryption process, the LPT and RPT block are rejoined. After that, the Final Permutation (FP) is applied to the combined block.
- Finally, we get the 64-bit ciphertext of the plaintext.



Decryption Step of the Algorithm

For decryption of the ciphertext, we use the same algorithm but in reverse order (step 4) of 16 round keys.

For better understanding of the algorithm, let's see modes of operation for the DES algorithm.

Modes of Operation For DES

There are the following five modes of operation that can be chosen:

1. **ECB (Electronic Codebook):** Each 64-bit block is encrypted and decrypted independently.
2. **CBC (Cipher Block Chaining):** In block chaining, each block depends on the previous one and uses an Initialization Vector (IV).
3. **CFB (Cipher Feedback):** The ciphertext that we get from the previous step becomes the input for the algorithm. The operation produces the pseudorandom output. The output that we get is XORed with the plaintext and generates the ciphertext for the next operation.
4. **OFB (Output Feedback):** It is just like CFB. Except that the encryption algorithm input is the output from the preceding DES.
5. **CTR (Counter):** Each plaintext block is XORed with an encrypted counter. After that, the counter is incremented for each subsequent block.

We conclude that the basic steps of the algorithm are:

- o First, we need to generate a secret **key** by using a **KeyGenerator**.
- o Create the two **Ciphers** one for encryption and the other for decryption. Remember that the key must be the same as we have specified in **Initialization Vector (IV)**.
- o At last, write and read the encrypted or decrypted data by using the **CipherOutputStream** and **CipherInputStream**.

Let's implement the DES algorithm in a Java program and see how data is encrypted and decrypted using the algorithm.

DesProgram.java

//Java classes that are mandatory to import for encryption and decryption process

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.AlgorithmParameterSpec;
import javax.crypto.Cipher;
import javax.crypto.CipherInputStream;
import javax.crypto.CipherOutputStream;
import javax.crypto.KeyGenerator;
```

```

import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
public class DesProgram
{
    //creating an instance of the Cipher class for encryption
    private static Cipher encrypt;
    //creating an instance of the Cipher class for decryption
    private static Cipher decrypt;
    //initializing vector
    private static final byte[] initialization_vector = { 22, 33, 11, 44, 55, 99, 66, 77 };
    //main() method
    public static void main(String[] args)
    {
        //path of the file that we want to encrypt
        String textFile = "C:/Users/Anubhav/Desktop/DemoData.txt";
        //path of the encrypted file that we get as output
        String encryptedData = "C:/Users/Anubhav/Desktop/encrypteddata.txt";
        //path of the decrypted file that we get as output
        String decryptedData = "C:/Users/Anubhav/Desktop/decrypteddata.txt";
        try
        {
            //generating keys by using the KeyGenerator class
            SecretKey scrkey = KeyGenerator.getInstance("DES").generateKey();
            AlgorithmParameterSpec aps = new IvParameterSpec(initialization_vector);
            //setting encryption mode
            encrypt = Cipher.getInstance("DES/CBC/PKCS5Padding");
            encrypt.init(Cipher.ENCRYPT_MODE, scrkey, aps);
            //setting decryption mode
            decrypt = Cipher.getInstance("DES/CBC/PKCS5Padding");
            decrypt.init(Cipher.DECRYPT_MODE, scrkey, aps);
            //calling encrypt() method to encrypt the file
            encryption(new FileInputStream(textFile), new FileOutputStream(encryptedData));
            //calling decrypt() method to decrypt the file
            decryption(new FileInputStream(encryptedData), new FileOutputStream(decryptedData));
            //prints the statement if the program runs successfully
            System.out.println("The encrypted and decrypted files have been created successfully.");
        }
        //catching multiple exceptions by using the | (or) operator in a single catch block
        catch (NoSuchAlgorithmException | NoSuchPaddingException | InvalidKeyException | InvalidAlgorithmParameterException | IOException e)
        {
            //prints the message (if any) related to exceptions
            e.printStackTrace();
        }
    }
    //method for encryption
    private static void encryption(InputStream input, OutputStream output)
    throws IOException

```



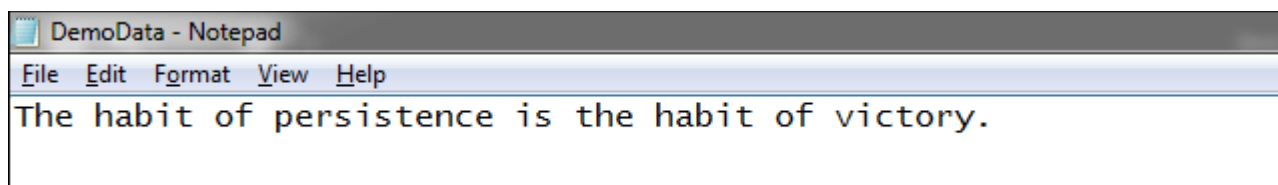
```

{
output = new CipherOutputStream(output, encrypt);
//calling the writeBytes() method to write the encrypted bytes to the file
writeBytes(input, output);
}
//method for decryption
private static void decryption(InputStream input, OutputStream output)
throws IOException
{
input = new CipherInputStream(input, decrypt);
//calling the writeBytes() method to write the decrypted bytes to the file
writeBytes(input, output);
}
//method for writing bytes to the files
private static void writeBytes(InputStream input, OutputStream output)
throws IOException
{
byte[] writeBuffer = new byte[512];
int readBytes = 0;
while ((readBytes = input.read(writeBuffer)) >= 0)
{
output.write(writeBuffer, 0, readBytes);
}
//closing the output stream
output.close();
//closing the input stream
input.close();
}
}

```

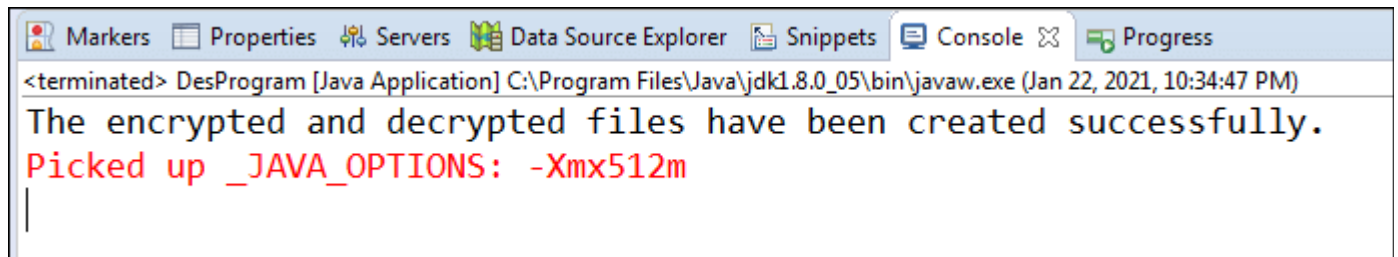
Before running the above program, you will have to do some changes in the program. First, you will have to create a text file that you want to encrypt. In our case, we have created a file with the name **DemoData.txt** and written the following text into it. You can write anything.

DemoData.txt



Let's run the above program and see the output.

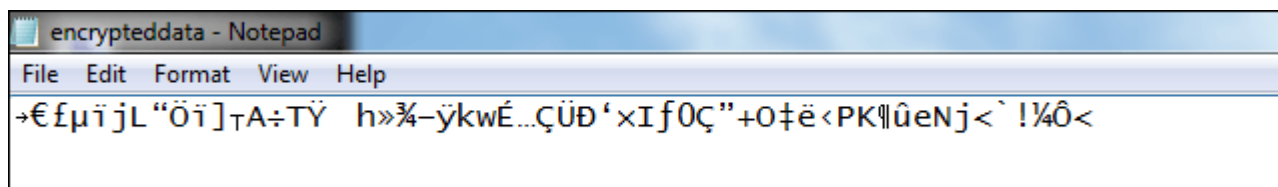
Output:



```
<terminated> DesProgram [Java Application] C:\Program Files\Java\jdk1.8.0_05\bin\javaw.exe (Jan 22, 2021, 10:34:47 PM)
The encrypted and decrypted files have been created successfully.
Picked up _JAVA_OPTIONS: -Xmx512m
```

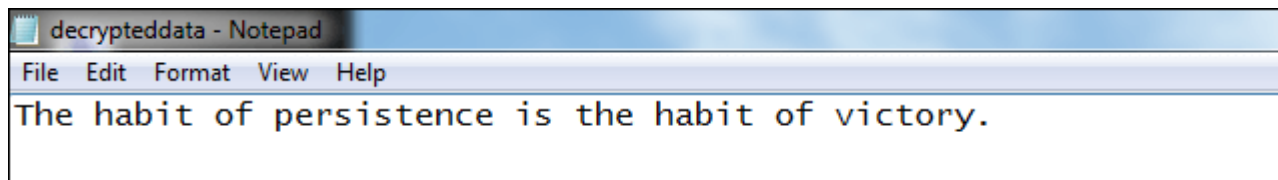
When we run the above program, it generates the two files **encrypteddata.txt** and **deecrypteddata.txt** at the specified location. Let's see what inside the encrypted and decrypted file.

encrypteddata.txt



```
encrypteddata - Notepad
File Edit Format View Help
→€fμījL“Öī]τA÷Tÿ h»¼-ÿkwÉ...ÇÜĐ‘×If0Ç”+O‡ë<PK¶ûeNj<`!¼ô<
```

deecrypteddata.txt



```
deecrypteddata - Notepad
File Edit Format View Help
The habit of persistence is the habit of victory.
```

We see that data is decrypted into the same text as we had written in the **DemoData.txt** file.

<https://www.javatpoint.com/java-code-for-des>

II. Java code for AES

AES 256 Encryption in Java

Security has become an important aspect nowadays. Java programming provides security for data transfer as well as communication between several nodes by supporting different encryption and hashing algorithms. In this section, we will discuss the **AES 256 encryption** algorithm and implement the logic in a Java program.

What is AES?

AES is an Advanced Encryption Standard algorithm. It is a type of symmetric, block cipher encryption and decryption algorithm. It works with key size 128, 192, and 256 bits. It uses a valid and similar secret key for both encryption and decryption.

In AES, the block cipher is used. It means that the data to be encrypted is converted into blocks for encryption. The original data value is encrypted using different bits of padding such as 128, 192, or 256 bits.

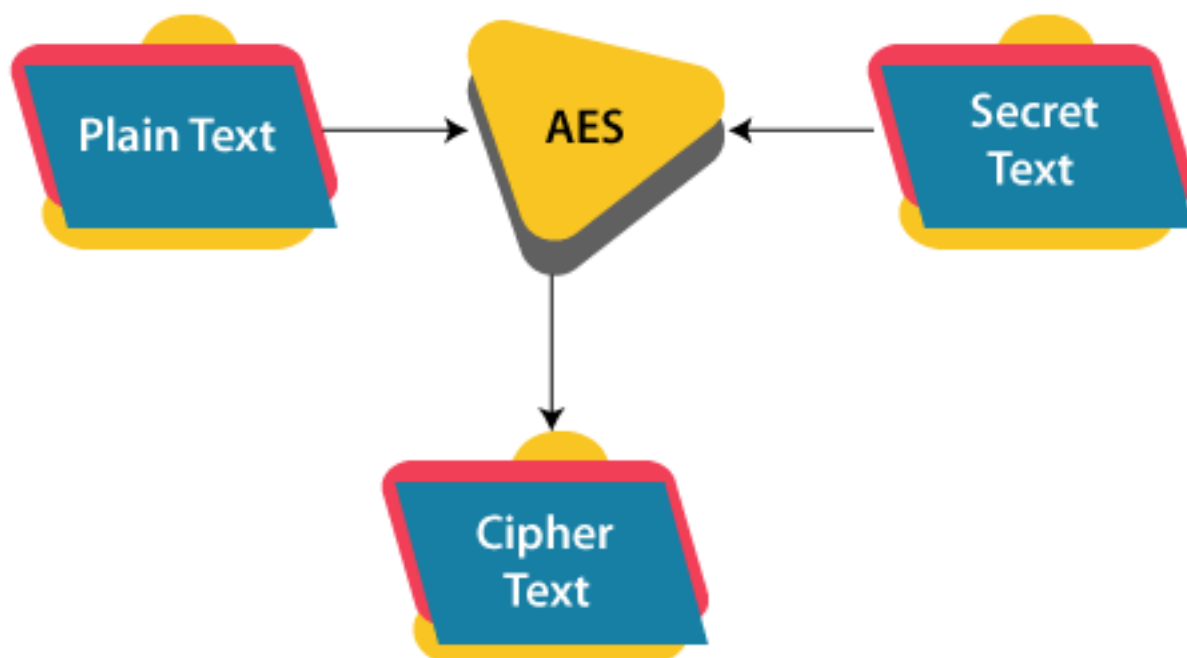
Advantages of AES

- o The encrypted data cannot be decrypted without a valid secret key.
- o AES is the most common security algorithm used worldwide for various purposes like wireless communication, financial transactions, encrypted data storage, etc.
- o The companies who want to transfer their data safely and without breaking it can always use the AES algorithm.

Disadvantages of AES

- o AES algorithm uses very simple algebraic formulae.
- o Each block is encrypted using a similar kind of encryption.
- o AES can be difficult to implement with the software.

AES 256 Encryption and Decryption



- o Using the AES encryption algorithm, a plain text message is converted into a cipher text with the help of a secret key that is only known to the sender and receiver of the message.
- o Encrypting or decrypting a message or a string is supported by Java Cryptographic Extension (JCE) framework in Java.
- o The Java Cryptographic Extension framework provides different packages for encryption and decryption.
 - o `java.security`
 - o `java.security.cert`
 - o `java.security.spec`
 - o `java.security.interfaces`
 - o `javax.crypto`
 - o `javax.crypto.spec`
 - o `javax.crypto.interfaces`
- o While decrypting a message, the reverse process of encryption is followed. It requires the value of the secret key in order to acquire the original message.
- o The Cipher class in Java is used for the encryption and decryption process. The `init()` method of the Cipher class initializes the cipher using the public key from the given transformation type.

Modes of Operation of AES Algorithm

There are the following six modes of operation in the AES algorithm:

1. ECB (Electronic Code Book):

It is the simplest mode among all. It divides the plaintext message into blocks of size 128 bits. Then these blocks are encrypted using the same key and algorithm. Hence, it generates the same cipher text for the same block every time. It is considered a weakness and therefore it is suggested not to use ECB for encryption.

2. CBC (Cipher Block Chaining):

CBC uses an **Initialization Vector** (IV) to improve the encryption. In CBC, the encryption is performed by XOR operation between the plaintext and IV. Then the cipher text is generated. It then uses the encryption result to XOR with the plain text until the last block.

3. CFB (Cipher FeedBack):

CFB can be used as a **stream cipher**. It encrypts the initialization vector (IV) first and then XOR with the plaintext to generate the cipher text. Then it encrypts the cipher text with the next plaintext block. In this mode, decryption can be performed in a parallel manner but encryption cannot be performed in a parallel manner.

4. OFB (Output FeedBack):

OFB can also be used as a stream cipher. It does not need padding data. First, the IV is encrypted and then the encryption result is XOR with the plaintext to generate the cipher text. Here, the IV cannot be encrypted or decrypted in a parallel manner.

5. CTR (Counter):

In CTR mode the encryption process is similar to OFB mode, the only difference is that it encrypts the counter value instead of IV.

It has two advantages, encryption or decryption can be performed in a parallel manner and the noise of one block does not affect another block.

6. GCM (Galois/Counter Mode):

GCM mode is an extended version of CTR mode. It was introduced by NIST. The GCM mode provides the cipher text as well as authentication tag after the encryption process.

In the following program, the AES/CBC/PKCS5Padding algorithm is used as it is popular and used in many projects.

Detail description

[Advanced Encryption Standard \(AES\)](#) is a specification for the encryption of electronic data established by the U.S National Institute of Standards and Technology (NIST) in 2001. AES is widely used today as it is a much stronger than DES and triple DES despite being harder to implement.

Points to remember

- AES is a block cipher.
- The key size can be 128/192/256 bits.
- Encrypts data in blocks of 128 bits each.

That means it takes 128 bits as input and outputs 128 bits of encrypted cipher text as output. AES relies on substitution-permutation network principle which means it is performed using a series of linked operations which involves replacing and shuffling of the input data.

Working of the cipher :

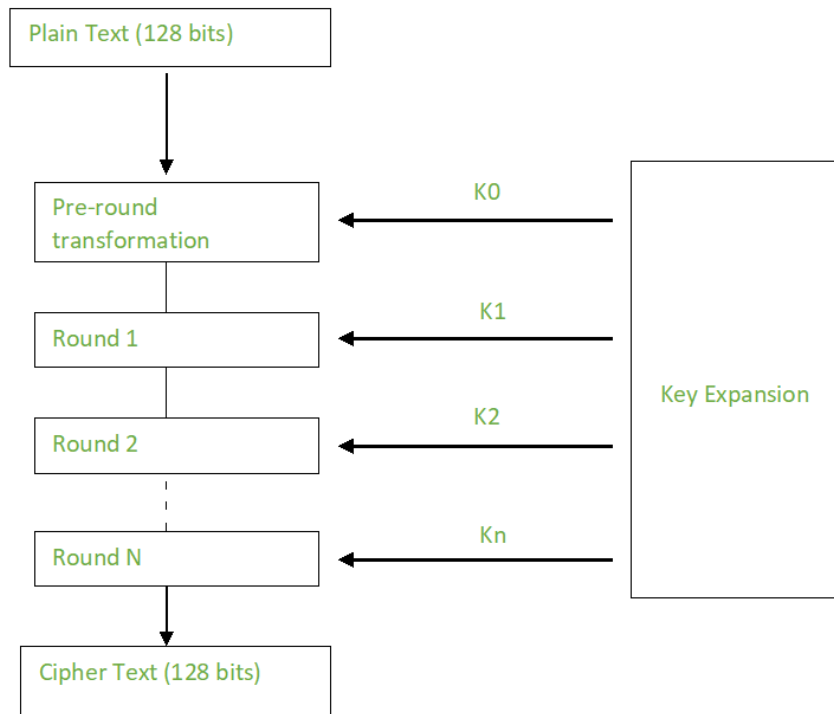
AES performs operations on bytes of data rather than in bits. Since the block size is 128 bits, the cipher processes 128 bits (or 16 bytes) of the input data at a time.

The number of rounds depends on the key length as follows :

- 128 bit key – 10 rounds
- 192 bit key – 12 rounds
- 256 bit key – 14 rounds

Creation of Round keys :

A Key Schedule algorithm is used to calculate all the round keys from the key. So the initial key is used to create many different round keys which will be used in the corresponding round of the encryption.



Encryption :

AES considers each block as a 16 byte (4 byte x 4 byte = 128) grid in a column major arrangement.

```

[ b0 | b4 | b8 | b12 |
  b1 | b5 | b9 | b13 |
  b2 | b6 | b10| b14 |
  b3 | b7 | b11| b15 ]
  
```

Each round comprises of 4 steps :

- SubBytes
- ShiftRows
- MixColumns
- Add Round Key

The last round doesn't have the MixColumns round.

The SubBytes does the substitution and ShiftRows and MixColumns performs the permutation in the algorithm.

SubBytes :

This step implements the substitution.

In this step each byte is substituted by another byte. Its performed using a lookup table also called the S-box. This substitution is done in a way that a byte is never substituted by itself and also not substituted by another byte which is a compliment of the current byte. The result of this step is a 16 byte (4 x 4) matrix like before.

The next two steps implement the permutation.

ShiftRows :

This step is just as it sounds. Each row is shifted a particular number of times.

- The first row is not shifted
- The second row is shifted once to the left.
- The third row is shifted twice to the left.
- The fourth row is shifted thrice to the left.

(A left circular shift is performed.)

| | | |
|---------------------------|----|---------------------------|
| [b0 b1 b2 b3] | | [b0 b1 b2 b3] |
| b4 b5 b6 b7 | -> | b5 b6 b7 b4 |
| b8 b9 b10 b11 | | b10 b11 b8 b9 |
| [b12 b13 b14 b15] | | [b15 b12 b13 b14] |

MixColumns :

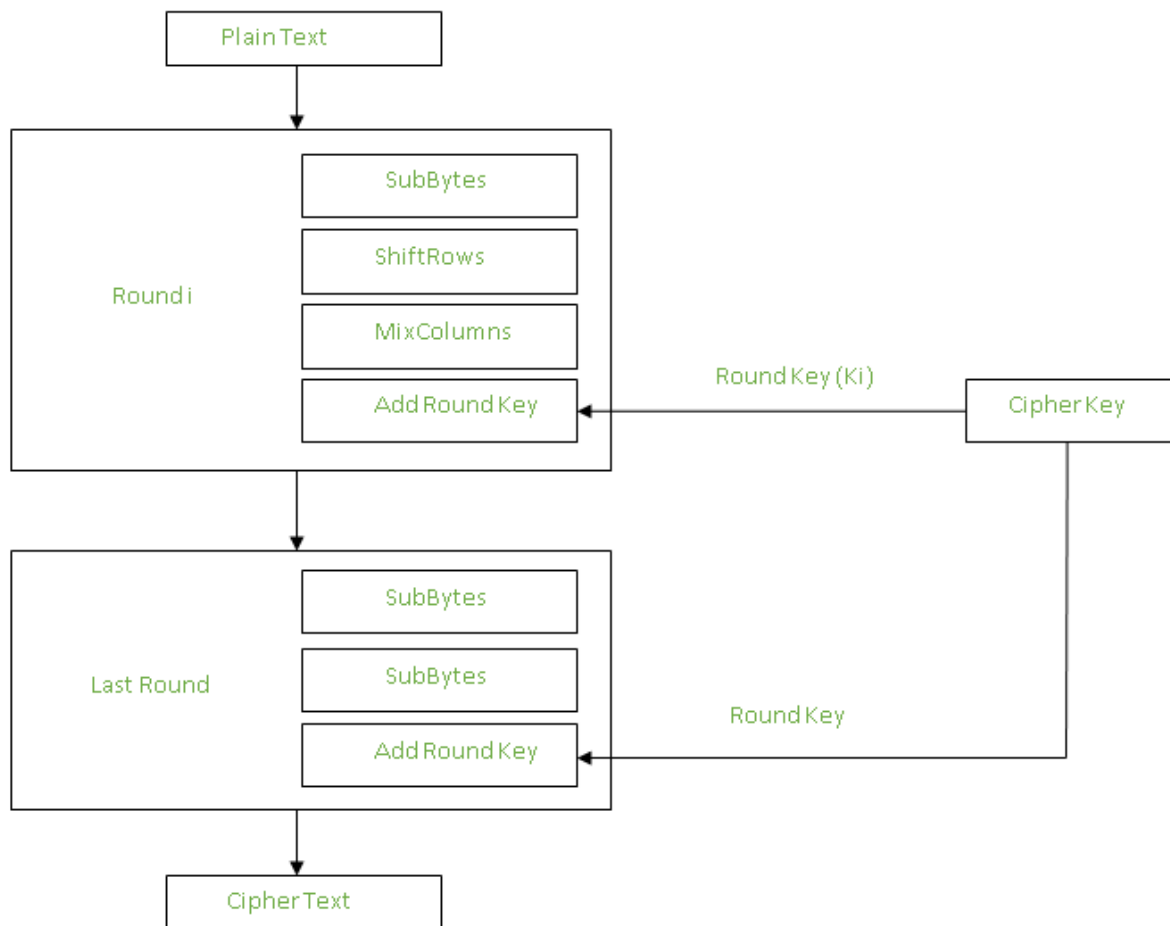
This step is basically a matrix multiplication. Each column is multiplied with a specific matrix and thus the position of each byte in the column is changed as a result.

This step is skipped in the last round.

| | | | |
|--------|---|-------------|--------|
| [c0] | | [2 3 1 1] | [b0] |
| c1 | = | 1 2 3 1 | b1 |
| c2 | | 1 1 2 3 | b2 |
| [c3] | | [3 1 1 2] | [b3] |

Add Round Keys :

Now the resultant output of the previous stage is XOR-ed with the corresponding round key. Here, the 16 bytes is not considered as a grid but just as 128 bits of data.



After all these rounds 128 bits of encrypted data is given back as output. This process is repeated until all the data to be encrypted undergoes this process.

Decryption :

The stages in the rounds can be easily undone as these stages have an opposite to it which when performed reverts the changes. Each 128 blocks goes through the 10, 12 or 14 rounds depending on the key size.

The stages of each round in decryption is as follows :

- Add round key
- Inverse MixColumns
- ShiftRows
- Inverse SubByte

The decryption process is the encryption process done in reverse so i will explain the steps with notable differences.

Inverse MixColumns :

This step is similar to the MixColumns step in encryption, but differs in the matrix used to carry out the operation.

$$\begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$$

| | | |
|--------|----------------|--------|
| b2 | 13 9 14 11 | c2 |
| [b3] | [11 13 9 14] | [c3] |

Inverse SubBytes :

Inverse S-box is used as a lookup table and using which the bytes are substituted during decryption.

Summary :

AES instruction set is now integrated into the CPU (offers throughput of several GB/s) to improve the speed and security of applications that use AES for encryption and decryption. Even though it's been 20 years since its introduction we have failed to break the AES algorithm as it is infeasible even with the current technology. Till date the only vulnerability remains in the implementation of the algorithm.

AES-256 Encryption and Decryption Java Program

AESExample.java

```
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.KeySpec;
import java.util.Base64;
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
public class AESExample
{
    /* Private variable declaration */
    private static final String SECRET_KEY = "123456789";
    private static final String SALTVALUE = "abcdefg";

    /* Encryption Method */
    public static String encrypt(String strToEncrypt)
    {
        try
        {
            /* Declare a byte array. */
            byte[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
            IvParameterSpec ivspec = new IvParameterSpec(iv);
```

```

    /* Create factory for secret keys. */
    SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
    /* PBEKeySpec class implements KeySpec interface. */
    KeySpec spec = new PBEKeySpec(SECRET_KEY.toCharArray(), SALTVALUE.getBytes(), 65536, 25
6);
    SecretKey tmp = factory.generateSecret(spec);
    SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivspec);
    /* Retrurns encrypted value. */
    return Base64.getEncoder()
.encodeToString(cipher.doFinal(strToEncrypt.getBytes(StandardCharsets.UTF_8)));
}
catch (InvalidAlgorithmParameterException | InvalidKeyException | NoSuchAlgorithmException | InvalidKeySpecException | BadPaddingException | IllegalBlockSizeException | NoSuchPaddingException e)
{
    System.out.println("Error occured during encryption: " + e.toString());
}
return null;
}

/* Decryption Method */
public static String decrypt(String strToDecrypt)
{
    try
    {
        /* Declare a byte array. */
        byte[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
        IvParameterSpec ivspec = new IvParameterSpec(iv);
        /* Create factory for secret keys. */
        SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
        /* PBEKeySpec class implements KeySpec interface. */
        KeySpec spec = new PBEKeySpec(SECRET_KEY.toCharArray(), SALTVALUE.getBytes(), 6553
6, 256);
        SecretKey tmp = factory.generateSecret(spec);
        SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
        cipher.init(Cipher.DECRYPT_MODE, secretKey, ivspec);
        /* Retrurns decrypted value. */
        return new String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
    }
    catch (InvalidAlgorithmParameterException | InvalidKeyException | NoSuchAlgorithmException | InvalidKeySpecException | BadPaddingException | IllegalBlockSizeException | NoSuchPaddingException e)
    {
        System.out.println("Error occured during decryption: " + e.toString());
    }
}

```

```

return null;
}
/* Driver Code */
public static void main(String[] args)
{
    /* Message to be encrypted. */
    String originalval = "AES Encryption";
    /* Call the encrypt() method and store result of encryption. */
    String encryptedval = encrypt(originalval);
    /* Call the decrypt() method and store result of decryption. */
    String decryptedval = decrypt(encryptedval);
    /* Display the original message, encrypted message and decrypted message on the console. */
    System.out.println("Original value: " + originalval);
    System.out.println("Encrypted value: " + encryptedval);
    System.out.println("Decrypted value: " + decryptedval);
}
}

```

Output:

```

Original value: AES Encryption
Encrypted value: V5E9I52IxxMaW4+hJhl56g==
Decrypted value: AES Encryption

```

In the above Java program, the ***AESExample*** class defines two methods, ***encrypt()*** that implements the AES-256 encryption algorithm and ***decrypt()*** that implements the AES-256 decryption algorithm. And lastly, the driver method gives a call to both the methods and displays the result on the console.

In this article, we have discussed the AES 256 encryption algorithm in Java, its modes of operations with its implementation as well as its pros and cons.

III. Practice with Symmetric Key Cryptography in Java: DES, AES

Download code here:

[https://drive.google.com/drive/folders/1alCX6HtDSPkeXTk65Bd4tgWvJ39_KVPk?usp=share link](https://drive.google.com/drive/folders/1alCX6HtDSPkeXTk65Bd4tgWvJ39_KVPk?usp=share_link)

(<https://stackjava.com/demo/code-java-vi-du-ma-hoa-giai-ma-voi-aes.html>

<https://stackjava.com/demo/code-java-vi-du-ma-hoa-giai-ma-voi-des.html>)