EBook Gratis

APRENDIZAJE cryptography

Free unaffiliated eBook created from
Stack Overflow contributors.

#cryptograp

hy

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: cryptography

It is an unofficial and free cryptography ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official cryptography.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con la criptografía

Observaciones

La criptografía moderna es la piedra angular de la seguridad informática y de las comunicaciones. Su fundamento se basa en conceptos matemáticos como la teoría de los números, la teoría de la complejidad computacional y la teoría de la probabilidad.

La criptografía se ocupa de la obtención de datos digitales. Se refiere al diseño de mecanismos basados en algoritmos matemáticos. El objetivo principal del uso de la criptografía es proporcionar los cuatro servicios fundamentales de seguridad de la información; Confidencialidad, no repudio, autenticación e integridad de los datos.

Examples

Integridad validada - Clave simétrica - Ejemplo de cifrado y descifrado utilizando Java

El cifrado se utiliza para transformar los datos en su formato original (p. Ej., El contenido de una carta, parte de las Credenciales para autorizar una transacción financiera) a algo que no puede ser reconstruido fácilmente por alguien que no esté destinado a formar parte de la conversación.

Básicamente, el cifrado se utiliza para evitar las escuchas ilegales entre dos entidades (individuos o un grupo).

En el caso de cifrado simétrico, tanto el remitente como el receptor (Ej .: Alice, Bob) deben usar el mismo algoritmo de cifrado (generalmente uno estandarizado) y la misma clave de cifrado (conocida solo por los dos).

http://docs.oracle.com/javase/1.5.0/docs/guide/security/jce/JCERefGuide.html#Examples

enlaces relacionados

- https://en.wikipedia.org/wiki/History_of_cryptography
- https://en.wikipedia.org/wiki/Criptografía

```
package com.example.so.documentation.cryptography;

import java.nio.charset.Charset;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.AlgorithmParameterSpec;
import java.util.StringTokenizer;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
```

```
import javax.crypto.KeyGenerator;
import javax.crypto.Mac;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;


/**
 *
 * <p> Encryption is used to transform data in its orignal format (Eg: The contents of a
letter, Credentials part of authorizing a financial transaction) to something that
 * cannot be easily reconstructed by anyone who is not intended to be part of the
conversation.  </p>
 * <p> Basically encryption is used to prevent eavesdropping between any two entities
 * (individuals or a group). </p>
 *
 * <p> In case of symmetric encryption, both the sender and receiver (Eg: Alice, Bob) must use
the same encryption algorithm (generally a standardised one)
 * and the same encryption key (known only to the two of them). </p>
 *
 * <p> http://docs.oracle.com/javase/1.5.0/docs/guide/security/jce/JCERefGuide.html#Examples
</p>
 *
 * <p> Related Links </p>
 * <ul>
 *     <li>https://en.wikipedia.org/wiki/History_of_cryptography</li>
 *     <li>https://en.wikipedia.org/wiki/Cryptography</li>
 * </ul>
 *
 * <pre>
 *         ChangeLog : 2016-09-24
 *         1. The modified encrypted text is now reflected correctly in the log and also
updated same in javadoc comment.
 * </pre>
 * @author Ravindra HV (with inputs w.r.t integrity check from
ArtjomB[http://stackoverflow.com/users/1816580/artjom-b])
 * @since (30 July 2016)
 * @version 0.3
 *
 */
public class IntegrityValidatedSymmetricCipherExample {

    /**
     * <p>https://en.wikipedia.org/wiki/Advanced_Encryption_Standard</p>
     */
    private static final String SYMMETRIC_ENCRYPTION_ALGORITHM_NAME = "AES"; // The current
standard encryption algorithm (as of writing)

    /**
     * <p>Higher the number, the better</p>
     * <p>Encryption is performed on chunks of data defined by the key size</p>
     * <p>Higher key sizes may require modification to the JDK (Unlimited Strength
Cryptography)</p>
     *
     */
    private static final int SYMMETRIC_ENCRYPTION_KEY_SIZE = 128; // lengths can be 128, 192
and 256

    /**
```

```
     * <p>
     *         A transformation defines in what manner the encryption should be performed.
     * </p>
     * <p>
            Eg: Whether there is any link between two chunks of encrypted data (CBC) or what
should happen
     *         if there is a mismatch between the key-size and the data length.       *
     * </p>
     *
     * <p> https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation </p>
     */
    private static final String SYMMETRIC_ENCRYPTION_TRANSFORMATION = "AES/CBC/PKCS5Padding";
    private static final Charset CHARSET_INSTANCE_UTF8 = Charset.forName("UTF-8");


    private static final int AES_IV_KEY_SIZE = 128; // for AES, iv key size is fixed at 128
independent of key-size

    private static final String MAC_ALGORITHM_NAME__HMAC_SHA256 = "HmacSHA256";
    private static final String HASH_FIELDS_SEPARATOR = "|" ;




    /**
     * @param args
     * <p>Sample output.</p>
     * <pre>
Encrypted, Base64 encoded text
:W1DePjeYMlI6xmyq9jr+cw==|55F80F4C2987CC143C69563025FACE22|GLR3T8GdcocpsTM1qSXp5jLsNx6QRK880BtgnV1jFg0=

Decrypted text :helloworld
Encrypted, Base64 encoded text -
v2:1XX/A9BO1Cp8mK+SHh9iHA==|B8294AC9967BB57D714ACCB3EE5710BD|TnjdaWbvp+H6yCbAAQFMkWNixeW8VwmW48YlKA/AAy

Decrypted text  - v2:helloworld
Encrypted, Base64 encoded text - v3
(original):EU4+rAZ2vOKtoSDiDPcO+A==|AEEB8DD341D8D9CD2EDFA05A4595EBD2|7anESSSJf1dHobS5tDdQ1mCNkFcIgCvtNC

Encrypted, Base64 encoded text - v3
(modified):FU4+rAZ2vOKtoSDiDPcO+A==|AEEB8DD341D8D9CD2EDFA05A4595EBD2|7anESSSJf1dHobS5tDdQ1mCNkFcIgCvtNC

Error : Integrity check failed
Exception in thread "main" java.lang.RuntimeException: Error : Integrity check failed
    at
com.example.so.documentation.cryptography.IntegrityValidatedSymmetricCipherExampleThree.decrypt(Integri

    at
com.example.so.documentation.cryptography.IntegrityValidatedSymmetricCipherExampleThree.main(IntegrityV

     * </pre>
     */
    public static void main(String[] args) {

        /*
         * EncryptionKey : Shared secret between receiver and sender (who generates the
password and how its shared depends on the purpose)
         * This program generates a new one every time its run !
         * Normally it would be generated once and then be stored somewhere (Eg: In a JCEKS
keystore file).
         */
```

```
        byte[] generatedSharedSecret = secretKeyGeneratorUtility();
        byte[] generatedSharedHMACKey = secretKeyGeneratorUtility();
        String plainText = "helloworld";

        String encryptedText = encrypt(plainText, generatedSharedSecret,
generatedSharedHMACKey);
        System.out.println("Encrypted, Base64 encoded text :"+encryptedText);
        String decryptedText = decrypt(encryptedText, generatedSharedSecret,
generatedSharedHMACKey);
        System.out.println("Decrypted text :"+decryptedText);

        String encryptedTextTwo = encrypt(plainText, generatedSharedSecret,
generatedSharedHMACKey);
        System.out.println("Encrypted, Base64 encoded text - v2:"+encryptedTextTwo);
        String decryptedTextTwo = decrypt(encryptedTextTwo, generatedSharedSecret,
generatedSharedHMACKey);
        System.out.println("Decrypted text  - v2:"+decryptedTextTwo);

        String encryptedTextThree = encrypt(plainText, generatedSharedSecret,
generatedSharedHMACKey);
        System.out.println("Encrypted, Base64 encoded text - v3
(original):"+encryptedTextThree);
        char[] encryptedTextThreeChars = encryptedTextThree.toCharArray();
        encryptedTextThreeChars[0] = (char) ((encryptedTextThreeChars[0])+1);
        String encryptedTextThreeModified = new String(encryptedTextThreeChars);
        System.out.println("Encrypted, Base64 encoded text - v3
(modified):"+encryptedTextThreeModified);

        String decryptedTextThree = decrypt(encryptedTextThreeModified, generatedSharedSecret,
generatedSharedHMACKey);
        System.out.println("Decrypted text  - v3:"+decryptedTextThree);

    }


    public static String encrypt(String plainText, byte[] key, byte[] hmacKey) {

        byte[] plainDataBytes = plainText.getBytes(CHARSET_INSTANCE_UTF8);
        byte[] iv = initializationVectorGeneratorUtility();
        byte[] encryptedDataBytes = encrypt(plainDataBytes, key, iv);

        String initializationVectorHex = DatatypeConverter.printHexBinary(iv);
        String encryptedBase64EncodedString =
DatatypeConverter.printBase64Binary(encryptedDataBytes); // Generally the encrypted data is
encoded in Base64 or hexadecimal encoding for ease of handling.
        String hashInputString = encryptedBase64EncodedString + HASH_FIELDS_SEPARATOR +
initializationVectorHex + HASH_FIELDS_SEPARATOR;
        String hashedOutputString =
DatatypeConverter.printBase64Binary(messageHashWithKey(hmacKey,
hashInputString.getBytes(CHARSET_INSTANCE_UTF8)));
        String encryptionResult = hashInputString + hashedOutputString;
        return encryptionResult;
    }

    public static byte[] encrypt(byte[] plainDataBytes, byte[] key, byte[] iv) {
        byte[] encryptedDataBytes = encryptOrDecrypt(plainDataBytes, key, iv, true);
        return encryptedDataBytes;
    }


    public static String decrypt(String cipherInput, byte[] key, byte[] hmacKey) {
```

```
        StringTokenizer stringTokenizer = new StringTokenizer(cipherInput,
HASH_FIELDS_SEPARATOR);

        String encryptedString = stringTokenizer.nextToken();
        String initializationVectorHex = stringTokenizer.nextToken();
        String hashedString = stringTokenizer.nextToken();

        String hashInputString = encryptedString + HASH_FIELDS_SEPARATOR +
initializationVectorHex + HASH_FIELDS_SEPARATOR;
        String hashedOutputString =
DatatypeConverter.printBase64Binary(messageHashWithKey(hmacKey,
hashInputString.getBytes(CHARSET_INSTANCE_UTF8)));

        if( hashedString.equals(hashedOutputString) == false ) {
            String message = "Error : Integrity check failed";
            System.out.println(message);
            throw new RuntimeException(message);
        }

        byte[] encryptedDataBytes = DatatypeConverter.parseBase64Binary(encryptedString); //
The Base64 encoding must be reversed so as to reconstruct the raw bytes.
        byte[] iv = DatatypeConverter.parseHexBinary(initializationVectorHex);
        byte[] plainDataBytes = decrypt(encryptedDataBytes, key, iv);
        String plainText = new String(plainDataBytes, CHARSET_INSTANCE_UTF8);
        return plainText;
    }

    public static byte[] decrypt(byte[] encryptedDataBytes, byte[] key, byte[] iv) {
        byte[] decryptedDataBytes = encryptOrDecrypt(encryptedDataBytes, key, iv, false);
        return decryptedDataBytes;
    }


    public static byte[] encryptOrDecrypt(byte[] inputDataBytes, byte[] key, byte[] iv,
boolean encrypt) {
        byte[] resultDataBytes = null;

        // Exceptions, if any, are just logged to console for this example.
        try {
            Cipher cipher = Cipher.getInstance(SYMMETRIC_ENCRYPTION_TRANSFORMATION);
            SecretKey secretKey = new SecretKeySpec(key, SYMMETRIC_ENCRYPTION_ALGORITHM_NAME);
            AlgorithmParameterSpec algorithmParameterSpec = new IvParameterSpec(iv);
            if(encrypt) {
                cipher.init(Cipher.ENCRYPT_MODE, secretKey, algorithmParameterSpec);
            }
            else {
                cipher.init(Cipher.DECRYPT_MODE, secretKey, algorithmParameterSpec);
            }

            resultDataBytes = cipher.doFinal(inputDataBytes); // In relative terms, invoking
do-final in one go is fine as long as the input size is small.
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (NoSuchPaddingException e) {
            e.printStackTrace();
        } catch (InvalidKeyException e) {
            e.printStackTrace();
        } catch (IllegalBlockSizeException e) {
            e.printStackTrace();
        } catch (BadPaddingException e) {
            e.printStackTrace();
```

```
        } catch (InvalidAlgorithmParameterException e) {
            e.printStackTrace();
        }

        return resultDataBytes;
    }


    private static byte[] secretKeyGeneratorUtility() {
        byte[] keyBytes = null;

        try {
            KeyGenerator keyGenerator =
KeyGenerator.getInstance(SYMMETRIC_ENCRYPTION_ALGORITHM_NAME);
            keyGenerator.init(SYMMETRIC_ENCRYPTION_KEY_SIZE);
            SecretKey secretKey = keyGenerator.generateKey();
            keyBytes = secretKey.getEncoded();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }

        return keyBytes;

    }


    /**
     * <p> InitialVector : Helps in avoiding generating the same encrypted result, even when
the same encryption - algorithm and key are used. </p>
     * <p> Since this is also required to be known to both sender and receiver, its either
based on some convention or is part of the cipher-text transmitted.</p>
     * <p> https://en.wikipedia.org/wiki/Initialization_vector </p>
     * @return
     */
    private static byte[] initializationVectorGeneratorUtility() {
        byte[] initialVectorResult = null;

        try {
            KeyGenerator keyGenerator =
KeyGenerator.getInstance(SYMMETRIC_ENCRYPTION_ALGORITHM_NAME);
            keyGenerator.init(AES_IV_KEY_SIZE);
            SecretKey secretKey = keyGenerator.generateKey();
            initialVectorResult = secretKey.getEncoded();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }

        return initialVectorResult;
    }


    private static byte[] messageHashWithKey(byte[] key, byte[] data) { // byte[] iv,
        byte[] hmac = null;

        try {
            Mac mac = Mac.getInstance(MAC_ALGORITHM_NAME__HMAC_SHA256);
            SecretKeySpec secretKeySpec = new SecretKeySpec(key,
MAC_ALGORITHM_NAME__HMAC_SHA256);
            //AlgorithmParameterSpec algorithmParameterSpec = new IvParameterSpec(iv);
            mac.init(secretKeySpec); // algorithmParameterSpec
            hmac = mac.doFinal(data);
```

```
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (InvalidKeyException e) {
            e.printStackTrace();
        } /*catch (InvalidAlgorithmParameterException e) {
            e.printStackTrace();
        }*/

        return hmac;

    }

}
```

Introducción

La criptografía es la ciencia del uso de construcciones matemáticas (códigos) para hacer que la comunicación sea segura. El campo de la criptografía es un subconjunto del campo de la seguridad de la información.

Hay muchas operaciones criptográficas posibles; Algunos ejemplos más conocidos son:

- **Cifrado:** transformar un mensaje de texto simple en un mensaje de texto cifrado para que el mensaje permanezca *confidencial*
- **Descifrado:** transformar un mensaje de texto cifrado de nuevo en un mensaje de texto simple
- **Hash seguro:** realizar una compresión irreversible (unidireccional) para crear una representación computacionalmente distinta y de tamaño estático para un mensaje específico.

La criptografía se basa en las matemáticas, y la aritmética se utiliza con frecuencia en algoritmos relacionados con la criptografía. Existe un pequeño subconjunto de primitivas, esquemas y protocolos que utilizan los desarrolladores. Los desarrolladores generalmente no implementan los algoritmos en sí mismos, sino que usan los esquemas y protocolos proporcionados por las API criptográficas y los tiempos de ejecución.

Una **primitiva** podría ser un cifrado de bloque como AES. Una primitiva es cualquier algoritmo que se utiliza como bloque de construcción para un esquema criptográfico. Un **esquema** es, por ejemplo, un *modo de operación de* cifrado por bloques, como CBC o GCM. Uno o más esquemas criptográficos pueden constituir un protocolo criptográfico. Un **protocolo** como TLS usa muchos esquemas criptográficos, pero también técnicas de codificación / decodificación de mensajes, orden de mensajes, condiciones de uso, etc. Las API criptográficas de bajo nivel solo proporcionan acceso directo a primitivas, mientras que las API de alto nivel pueden ofrecer acceso a implementaciones de protocolos completos.

Los mensajes se han cifrado y descifrado a mano desde que se inventó la palabra escrita. Los dispositivos mecánicos se han utilizado al menos desde la antigua sociedad griega. Este tipo de criptografía se conoce como *criptografía clásica* . Muchas introducciones a la criptografía comienzan con la criptografía clásica, ya que es relativamente fácil de analizar. Sin embargo, los algoritmos clásicos no se adhieren a la seguridad requerida de las construcciones modernas y, a

menudo, son fáciles de romper. Ejemplos de esquemas clásicos son el César y el Vigenère. El dispositivo mecánico más conocido es, sin duda, la máquina de codificación Enigma.

La criptografía moderna se basa en la ciencia, principalmente matemática y teoría de números / grupos. Se trata de algoritmos mucho más intrincados y tamaños de clave. Estos solo pueden ser manejados eficientemente por dispositivos informáticos. Por esta razón, la criptografía moderna utiliza principalmente la entrada y salida orientada a bytes. Esto significa que los mensajes deben convertirse a binarios y viceversa antes de que puedan ser transformados por cualquier implementación de un algoritmo criptográfico. Esto significa que los mensajes (de texto) deben transformarse utilizando la codificación de caracteres antes de ser cifrados.

Las formas de *codificación* de *caracteres* de los mensajes de texto son UTF-8. Los mensajes estructurados pueden codificarse utilizando ASN.1 / DER o representaciones XML *canónicas* , o cualquier número de técnicas patentadas. A veces, la salida binaria también debe transformarse de nuevo en texto. En este caso, se puede utilizar un esquema de *codificación* como base 64 o hexadecimals para representar los datos binarios dentro del texto.

La criptografía es notoriamente difícil de acertar. Los desarrolladores solo deben usar construcciones que entiendan completamente. Si es posible, un desarrollador debe usar un protocolo de nivel superior como TLS para crear seguridad de transporte. No hay una posibilidad práctica de crear un algoritmo, esquema o protocolo seguro sin educación formal o experiencia extensa. Copiar / pegar ejemplos de Internet no es probable que conduzca a soluciones seguras e incluso puede resultar en la pérdida de datos.

Lea Empezando con la criptografía en línea:
https://riptutorial.com/es/cryptography/topic/3408/empezando-con-la-criptografia

Capítulo 2: Cifra de juego

Introducción

El cifrado de encriptación de múltiples letras más conocido es Playfair, que trata los digramas en el texto plano como unidades individuales y traduce estas unidades en digramas de texto cifrado. El algoritmo de Playfair se basa en el uso de una matriz de letras de 5 x 5 construidas con una palabra clave.

Examples

Ejemplo de cifrado de cifrado de Playfair junto con el cifrado y la regla de descifrado

Una palabra clave es "MONARQUÍA", entonces la matriz se verá como

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

La matriz se construye rellenando las letras de la palabra clave (menos los duplicados) de izquierda a derecha y de arriba a abajo, y luego rellenando el resto de la matriz con las letras restantes en orden alfabético. El texto simple se cifra dos letras a la vez, de acuerdo con las siguientes reglas:

1. Tome los caracteres en el texto (plano / cifrado) y haga un grupo de dos caracteres. Si el número de caracteres en el texto es impar, agregue una letra de relleno (usualmente usamos 'x').
   Texto = "HOLA" entonces sería un grupo como
   El | LL | BUEY

2. Dos letras de texto sin formato que caen en la misma fila de la matriz se reemplazan con la letra de la derecha, y el primer elemento de la fila sigue circularmente el último. Por ejemplo, ar está encriptado como RM.

3. Dos letras de texto sin formato que caen en la misma columna son reemplazadas por la siguiente letra, y el elemento superior de la columna sigue circularmente la última. Por ejemplo, mu está encriptado como CM.

4. De lo contrario, cada letra de texto simple en un par se reemplaza por la letra que se encuentra en su propia fila y la columna ocupada por la otra letra de texto simple. Por lo tanto, hs se convierte en BP y ea se convierte en IM (o JM, según lo desee el codificador).

A continuación se presenta un programa Java simple que implementa el cifrado de Playfair:

```
import java.util.Scanner;


public class Playfair {
public static void main(String[] args) {
    Scanner in=new Scanner(System.in);

    System.out.print("Enter keyword: ");
    String key=in.nextLine();
    System.out.print("Enter message to encrypt: ");
    String msg=in.nextLine();

    PFEncryption pfEncryption=new PFEncryption();
    pfEncryption.makeArray(key);
    msg=pfEncryption.manageMessage(msg);
    pfEncryption.doPlayFair(msg, "Encrypt");
    String en=pfEncryption.getEncrypted();
    System.out.println("Encrypting....\n\nThe encrypted text is: " + en);
    System.out.println("=======================================");
    pfEncryption.doPlayFair(en, "Decrypt");
    System.out.print("\nDecrypting....\n\nThe encrypted text is: " +
pfEncryption.getDecrypted());
}
}

class PFEncryption{

private char [][] alphabets= new char[5][5];
private char[] uniqueChar= new char[26];
private String ch="ABCDEFGHIKLMNOPQRSTUVWXYZ";
private String encrypted="";
private String decrypted="";

void makeArray(String keyword){
    keyword=keyword.toUpperCase().replace("J","I");
```

```
boolean present, terminate=false;
int val=0;
int uniqueLen;
for (int i=0; i<keyword.length(); i++){
    present=false;
    uniqueLen=0;
    if (keyword.charAt(i)!= ' '){
        for (int k=0; k<uniqueChar.length; k++){
            if (Character.toString(uniqueChar[k])==null){
                break;
            }
            uniqueLen++;
        }
        for (int j=0; j<uniqueChar.length; j++){
            if (keyword.charAt(i)==uniqueChar[j]){
                present=true;
            }
        }
        if (!present){
            uniqueChar[val]=keyword.charAt(i);
            val++;
        }
    }
    ch=ch.replaceAll(Character.toString(keyword.charAt(i)), "");
}

for (int i=0; i<ch.length(); i++){
    uniqueChar[val]=ch.charAt(i);
    val++;
}
val=0;

for (int i=0; i<5; i++){
    for (int j=0; j<5; j++){
        alphabets[i][j]=uniqueChar[val];
        val++;
        System.out.print(alphabets[i][j] + "\t");
    }
    System.out.println();
}
}

String manageMessage(String msg){
int val=0;
int len=msg.length()-2;
String newTxt="";
String intermediate="";
while (len>=0){
    intermediate=msg.substring(val, val+2);
    if (intermediate.charAt(0)==intermediate.charAt(1)){
        newTxt=intermediate.charAt(0) + "x" + intermediate.charAt(1);
        msg=msg.replaceFirst(intermediate, newTxt);
        len++;
    }
    len-=2;
    val+=2;
}

if (msg.length()%2!=0){
    msg=msg+'x';
    }
```

```
        return msg.toUpperCase().replaceAll("J","I").replaceAll(" ","");
    }

void doPlayFair(String msg, String tag){
    int val=0;
    while (val<msg.length()){
        searchAndEncryptOrDecrypt(msg.substring(val, val + 2), tag);
        val+=2;
    }
}

void searchAndEncryptOrDecrypt(String doubblyCh, String tag){
    char ch1=doubblyCh.charAt(0);
    char ch2=doubblyCh.charAt(1);
    int row1=0, col1=0, row2=0, col2=0;
    for (int i=0; i<5; i++){
        for (int j=0; j<5; j++){
            if (alphabets[i][j]==ch1){
                row1=i;
                col1=j;
            }else if (alphabets[i][j]==ch2){
                row2=i;
                col2=j;
            }
        }
    }
    if (tag=="Encrypt")
        encrypt(row1, col1, row2, col2);
    else if(tag=="Decrypt")
        decrypt(row1, col1, row2, col2);
}

void encrypt(int row1, int col1, int row2, int col2){
    if (row1==row2){
        col1=col1+1;
        col2=col2+1;
        if (col1>4)
            col1=0;
        if (col2>4)
            col2=0;

encrypted+=(Character.toString(alphabets[row1][col1])+Character.toString(alphabets[row1][col2]));

    }else if(col1==col2){
        row1=row1+1;
        row2=row2+1;
        if (row1>4)
            row1=0;
        if (row2>4)
            row2=0;

encrypted+=(Character.toString(alphabets[row1][col1])+Character.toString(alphabets[row2][col1]));

    }else{

encrypted+=(Character.toString(alphabets[row1][col2])+Character.toString(alphabets[row2][col1]));

    }
}

void decrypt(int row1, int col1, int row2, int col2){
```

```
    if (row1==row2){
        col1=col1-1;
        col2=col2-1;
        if (col1<0)
            col1=4;
        if (col2<0)
            col2=4;

decrypted+=(Character.toString(alphabets[row1][col1])+Character.toString(alphabets[row1][col2]));

    }else if(col1==col2){
        row1=row1-1;
        row2=row2-1;
        if (row1<0)
            row1=4;
        if (row2<0)
            row2=4;

decrypted+=(Character.toString(alphabets[row1][col1])+Character.toString(alphabets[row2][col1]));

    }else{

decrypted+=(Character.toString(alphabets[row1][col2])+Character.toString(alphabets[row2][col1]));

    }
}

String getEncrypted(){
    return encrypted;
}
String getDecrypted(){
    return decrypted;
}



}
```

Lea Cifra de juego en línea: https://riptutorial.com/es/cryptography/topic/8869/cifra-de-juego

Capítulo 3: Cifrado César

Introducción

Es el simple cifrado monoalfabético de cambio en el que cada letra se reemplaza por una posición de la letra 3 (cifrado César real) por delante utilizando el orden alfabético circular, es decir, la letra después de Z es A. Entonces, cuando codificamos HOLA MUNDIAL, el texto cifrado se convierte en KHOORZRUOG.

Examples

Introducción

El cifrado César es un método de cifrado clásico. Funciona desplazando los caracteres en una cierta cantidad. Por ejemplo, si elegimos un cambio de 3, A se convertirá en D y E se convertirá en H.

El siguiente texto ha sido encriptado usando un turno de 23.

```
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD
```

Implementacion Python

La forma ASCII

Esto cambia los caracteres pero no importa si el nuevo personaje no es una letra. Esto es bueno si desea usar puntuación o caracteres especiales, pero no necesariamente le dará letras solo como resultado. Por ejemplo, "z" cambia 3 veces a "}".

```
def ceasar(text, shift):
    output = ""
    for c in text:
        output += chr(ord(c) + shift)
    return output
```

ROT13

ROT13 es un caso especial de cifrado César, con un cambio de 13. Solo se cambian las letras, y los espacios en blanco y los caracteres especiales se dejan como están.

Lo interesante es que ROT13 es un cifrado recíproco: aplicar ROT13 dos veces le dará la entrada inicial. De hecho, 2 * 13 = 26, el número de letras en el alfabeto.

Como ROT13 no tiene una clave como parámetro de entrada, a menudo se ve más como un *algoritmo de codificación* o, más específicamente, un *algoritmo de ofuscación en* lugar de un cifrado.

ROT13 solo dificulta la lectura directa de mensajes y, por lo tanto, a menudo se usa para mensajes ofensivos o juegos de palabras de bromas. No proporciona ninguna seguridad computacional.

Una implementación de Java para cifrado César

Implementación de la cifra César.

- Esta implementación realiza la operación de cambio solo en alfabetos en mayúsculas y minúsculas y retiene los otros caracteres (como el espacio como está).
- El cifrado César no es seguro según los estándares actuales.
- ¡El siguiente ejemplo es solo para fines ilustrativos!
- Referencia: [https://en.wikipedia.org/wiki/Caesar_cipher◆ (https://en.wikipedia.org/wiki/Caesar_cipher)

```
package com.example.so.cipher;

/**
 * Implementation of the Caesar cipher.
 * <p>
 * <ul>
 * <li>This implementation performs the shift operation only on upper and lower
 * case alphabets and retains the other characters (such as space as-is).</li>
 * <li>The Caesar cipher is not secure as per current standards.</li>
 * <li>Below example is for illustrative purposes only !</li>
 * <li>Reference: https://en.wikipedia.org/wiki/Caesar_cipher</li>
 * </ul>
 * </p>
 *
 * @author Ravindra HV
 * @author Maarten Bodewes (beautification only)
 * @since 2016-11-21
 * @version 0.3
 *
 */
public class CaesarCipher {

    public static final char START_LOWER_CASE_ALPHABET = 'a'; // ASCII-97
    public static final char END_LOWER_CASE_ALPHABET = 'z';   // ASCII-122

    public static final char START_UPPER_CASE_ALPHABET = 'A'; // ASCII-65
    public static final char END_UPPER_CASE_ALPHABET = 'Z';   // ASCII-90

    public static final int ALPHABET_SIZE = 'Z' - 'A' + 1;    // 26 of course

    /**
     * Performs a single encrypt followed by a single decrypt of the Caesar
     * cipher, prints out the intermediate values and finally validates
     * that the decrypted plaintext is identical to the original plaintext.
     *
     * <p>
```

```
     * This method outputs the following:
     *
     * <pre>
     * Plaintext  : The quick brown fox jumps over the lazy dog
     * Ciphertext : Qeb nrfzh yoltk clu grjmp lsbo qeb ixwv ald
     * Decrypted  : The quick brown fox jumps over the lazy dog
     * Successful decryption: true
     * </pre>
     * </p>
     *
     * @param args (ignored)
     */
    public static void main(String[] args) {

        int shift = 23;
        String plainText = "The quick brown fox jumps over the lazy dog";

        System.out.println("Plaintext  : " + plainText);

        String ciphertext = caesarCipherEncrypt(plainText, shift);
        System.out.println("Ciphertext : " + ciphertext);

        String decrypted = caesarCipherDecrypt(ciphertext, shift);
        System.out.println("Decrypted  : " + decrypted);
        System.out.println("Successful decryption: "
                + decrypted.equals(plainText));
    }

    public static String caesarCipherEncrypt(String plaintext, int shift) {
        return caesarCipher(plaintext, shift, true);
    }

    public static String caesarCipherDecrypt(String ciphertext, int shift) {
        return caesarCipher(ciphertext, shift, false);
    }

    private static String caesarCipher(
            String input, int shift, boolean encrypt) {

        // create an output buffer of the same size as the input
        StringBuilder output = new StringBuilder(input.length());

        for (int i = 0; i < input.length(); i++) {

            // get the next character
            char inputChar = input.charAt(i);

            // calculate the shift depending on whether to encrypt or decrypt
            int calculatedShift = (encrypt) ? shift : (ALPHABET_SIZE - shift);

            char startOfAlphabet;
            if ((inputChar >= START_LOWER_CASE_ALPHABET)
                    && (inputChar <= END_LOWER_CASE_ALPHABET)) {

                // process lower case
                startOfAlphabet = START_LOWER_CASE_ALPHABET;
            } else if ((inputChar >= START_UPPER_CASE_ALPHABET)
                    && (inputChar <= END_UPPER_CASE_ALPHABET)) {

                // process upper case
                startOfAlphabet = START_UPPER_CASE_ALPHABET;
```

```
            } else {

                // retain all other characters
                output.append(inputChar);

                // and continue with the next character
                continue;
            }

            // index the input character in the alphabet with 0 as base
            int inputCharIndex =
                    inputChar - startOfAlphabet;

            // cipher / decipher operation (rotation uses remainder operation)
            int outputCharIndex =
                    (inputCharIndex + calculatedShift) % ALPHABET_SIZE;

            // convert the new index in the alphabet to an output character
            char outputChar =
                    (char) (outputCharIndex + startOfAlphabet);

            // add character to temporary-storage
            output.append(outputChar);
        }

        return output.toString();
    }
}
```

Salida del programa:

```
Plaintext  : The quick brown fox jumps over the lazy dog
Ciphertext : Qeb nrfzh yoltk clu grjmp lsbo qeb ixwv ald
Decrypted  : The quick brown fox jumps over the lazy dog
Successful decryption: true
```

Implementacion Python

El siguiente ejemplo de código implementa el cifrado César y muestra las propiedades del cifrado.

Maneja tanto caracteres alfanuméricos en mayúsculas como en minúsculas, dejando todos los demás caracteres como estaban.

Se muestran las siguientes propiedades del cifrado César:

- teclas débiles;
- espacio bajo llave;
- el hecho de que cada clave tiene una clave recíproca (inversa);
- la relación con ROT13;

También muestra lo siguiente - más genérico - nociones criptográficas:

- teclas débiles;
- la diferencia entre ofuscación (sin clave) y cifrado;

- Bruto forzando una llave;
- La integridad faltante del texto cifrado.

```
def caesarEncrypt(plaintext, shift):
    return caesarCipher(True, plaintext, shift)

def caesarDecrypt(ciphertext, shift):
    return caesarCipher(False, ciphertext, shift)

def caesarCipher(encrypt, text, shift):
    if not shift in range(0, 25):
        raise Exception('Key value out of range')

    output = ""
    for c in text:
        # only encrypt alphanumerical characters
        if c.isalpha():
            # we want to shift both upper- and lowercase characters
            ci = ord('A') if c.isupper() else ord('a')

            # if not encrypting, we're decrypting
            if encrypt:
                output += caesarEncryptCharacter(c, ci, shift)
            else:
                output += caesarDecryptCharacter(c, ci, shift)
        else:
            # leave other characters such as digits and spaces
            output += c
    return output

def caesarEncryptCharacter(plaintextCharacter, positionOfAlphabet, shift):
    # convert character to the (zero-based) index in the alphabet
    n = ord(plaintextCharacter) - positionOfAlphabet
    # perform the >positive< modular shift operation on the index
    # this always returns a value within the range [0, 25]
    # (note that 26 is the size of the western alphabet)
    x = (n + shift) % 26 # <- the magic happens here
    # convert the index back into a character
    ctc = chr(x + positionOfAlphabet)
    # return the result
    return ctc

def caesarDecryptCharacter(plaintextCharacter, positionOfAlphabet, shift):
    # convert character to the (zero-based) index in the alphabet
    n = ord(plaintextCharacter) - positionOfAlphabet
    # perform the >negative< modular shift operation on the index
    x = (n - shift) % 26
    # convert the index back into a character
    ctc = chr(x + positionOfAlphabet)
    # return the result
    return ctc

def encryptDecrypt():
    print '--- Run normal encryption / decryption'
    plaintext = 'Hello world!'
    key = 3 # the original value for the Caesar cipher
    ciphertext = caesarEncrypt(plaintext, key)
    print ciphertext
    decryptedPlaintext = caesarDecrypt(ciphertext, key)
```

```
    print decryptedPlaintext
encryptDecrypt()

print '=== Now lets show some cryptographic properties of the Caesar cipher'

def withWeakKey():
    print '--- Encrypting plaintext with a weak key is not a good idea'
    plaintext = 'Hello world!'
    # This is the weakest key of all, it does nothing
    weakKey = 0
    ciphertext = caesarEncrypt(plaintext, weakKey)
    print ciphertext # just prints out the plaintext
withWeakKey();

def withoutDecrypt():
    print '--- Do we actually need caesarDecrypt at all?'
    plaintext = 'Hello world!'
    key = 3 # the original value for the Caesar cipher
    ciphertext = caesarEncrypt(plaintext, key)
    print ciphertext
    decryptionKey = 26 - key; # reciprocal value
    decryptedPlaintext = caesarEncrypt(ciphertext, decryptionKey)
    print decryptedPlaintext # performed decryption
withoutDecrypt()


def punnify():
    print '--- ROT 13 is the Caesar cipher with a given, reciprocal, weak key: 13'
    # The key is weak because double encryption will return the plaintext
    def rot13(pun):
        return caesarEncrypt(pun, 13)

    print 'Q: How many marketing people does it take to change a light bulb?'
    obfuscated = 'N: V jvyy unir gb trg onpx gb lbh ba gung.'
    print obfuscated
    deobfuscated = rot13(obfuscated)
    print deobfuscated
    # We should not leak the pun, right? Lets obfuscate afterwards!
    obfuscatedAgain = rot13(deobfuscated)
    print obfuscatedAgain
punnify()

def bruteForceAndLength():
    print '--- Brute forcing is very easy as there are only 25 keys in the range [1..25]'
    # Note that AES-128 has 340,282,366,920,938,463,463,374,607,431,768,211,456 keys
    # and is therefore impossible to bruteforce (if the key is correctly generated)
    key = 10;
    plaintextToFind = 'Hello Maarten!'
    ciphertextToBruteForce = caesarEncrypt(plaintextToFind, key)
    for candidateKey in range(1, 25):
        bruteForcedPlaintext = caesarDecrypt(ciphertextToBruteForce, candidateKey)
        # lets assume the adversary knows 'Hello', but not the name
        if bruteForcedPlaintext.startswith('Hello'):
            print 'key value: ' + str(candidateKey) + ' gives : ' + bruteForcedPlaintext

    print '--- Length of plaintext usually not hidden'
    # Side channel attacks on ciphertext lengths are commonplace! Beware!
    if len(ciphertextToBruteForce) != len('Hello Stefan!'):
        print 'The name is not Stefan (but could be Stephan)'
bruteForceAndLength()
```

```
def manInTheMiddle():
    print '--- Ciphers are vulnerable to man-in-the-middle attacks'
    # Hint: do not directly use a cipher for transport security
    moneyTransfer = 'Give Maarten one euro'
    key = 1
    print moneyTransfer
    encryptedMoneyTransfer = caesarEncrypt(moneyTransfer, key)
    print encryptedMoneyTransfer
    # Man in the middle replaces third word with educated guess
    # (or tries different ciphertexts until success)
    encryptedMoneyTransferWords  = encryptedMoneyTransfer.split(' ');
    encryptedMoneyTransferWords[2] = 'ufo' # unidentified financial object
    modifiedEncryptedMoneyTransfer = ' '.join(encryptedMoneyTransferWords)
    print modifiedEncryptedMoneyTransfer
    decryptedMoneyTransfer = caesarDecrypt(modifiedEncryptedMoneyTransfer, key)
    print decryptedMoneyTransfer
manInTheMiddle()
```

Lea Cifrado César en línea: https://riptutorial.com/es/cryptography/topic/7504/cifrado-cesar

Creditos

S. No	Capítulos	Contributors
1	Empezando con la criptografía	Community, Maarten Bodewes, Rachel Gallen, Ravindra HV
2	Cifra de juego	Dipesh Poudel
3	Cifrado César	Community, Dipesh Poudel, Maarten Bodewes, Ravindra HV