Universidad Diego Portales

ESCUELA DE INFORMÁTICA &
TELECOMUNICACIONES

Laboratorio 1: Protección de mensajes con cifrado Vigenère.

Autores:
Valentina Mella
Valentina Diaz

Profesor:
Marcos Fontova

9 de abril 2025

Índice

1. Introducción

Con el avance en las tecnologias la seguridad es fundamental principalmente en las telecomunicaciones , en este informe demostraremos una forma de implemetar una clase BigViginere la cual funciona como un cifrador y decifrador de mesajes utilizando el abecedario, más números y símbolos, esto en una matriz bidireccional.Esta permite un cifrado más seguro y adecuado para entornos donde se requiere la protección de datos en comunicaciones digitales la cual ofrece una mayor seguridad al momento de enviar datos que puedan ser vulnerables en la transmicion de la información.Una gran ventaja en esta implementación es el manejo de claves bastantes extensas las cuales no se pueden representar como una sola variable lo cual hace mas efectiva su función. El objetivo de este informe es explicar como funciona BigViginere y el cifrado en el ambito de la seguridad de la información transmitida ademas de lo facil que es adaptarse a diferentes tipos de comunicación. El código se encuentra en el siguiente enlace:
`https://github.com/valeed28/CodLab1/blob/main/LAB_1%20EDA`

2. Implementación

Durante el laboratorio, el equipo aplico de manera equilibrada la clase "public BigVigenere", clase dedicada al cifrado de claves en la cual se abarcaron todos los métodos aplicados en el código. En este código, los atributos y constructores fueron indicados al inicio de la clase, formando la matriz Alphabet, la cual consta de 64 caracteres, incluyendo mayúsculas, minúsculas y números en la matriz. A su vez, se menciona la construcción del atributo Key, el cual se enfoca en transformar la clave ingresada por el usuario para luego aplicar el algoritmo de cifrado/descifrado.

```
public BigVigenere(String numericKey) {  1 usage
    this.key = convertirClave(numericKey);
    this.alphabet = generarMatrizAlphabet(); }

private int[] convertirClave(String claveKey) {  3 usages
    try {
        String[] partesClave = claveKey.split( regex: " ");
        int[] arreglo = new int[partesClave.length];
        for (int i = 0; i < partesClave.length; i++) {
            arreglo[i] = Integer.parseInt(partesClave[i]) % alphabet_size; }
        return arreglo; }
    catch (NumberFormatException e) {
        throw new IllegalArgumentException("Clave invalida,revise que solo haya ingresado numeros separados por espacios."); }
}

private char[][] generarMatrizAlphabet() {  2 usages
    char[][] matriz = new char[alphabet_size][alphabet_size];
    for (int fila = 0; fila < alphabet_size; fila++) {
        for (int col = 0; col < alphabet_size; col++) {
            matriz[fila][col] = character.charAt((fila + col) % alphabet_size);}
    } return matriz; }
```

Figura 1: Class BigVigenere

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z
2	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
3	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
4	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
5	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
6	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
7	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
8	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
9	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
10	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
11	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
12	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
13	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
14	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
15	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
16	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ
17	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O
18	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P
19	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q
20	S	T	Y	S	T	Y	S	T	Y	S	T	Y	S	T	Y	S	T	Y	S	T	Y	S	T	Y	S	T	Y
21	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S
22	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T
23	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U
24	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V
25	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W
26	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X
27	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y

Tablero Vigènere para el Alfabeto Español - C43S4RS ©

Figura 2: Matriz Modelo

3. Métodos

- El primer constructor que hacemos es un constructor class BigVigenere que solicita la clave al usuario y la convierte en un arreglo, genera una matriz del alfabeto correspondiente.
- BigVigenere(String numericKey): Cumple con la misma función del anterior, solo que este no solicita el o los datos al usuario.
- El método int[] convertirClave(String claveKey) convierte una cadena de números separados por espacios en un arreglo de enteros, donde cada valor se reduce al tamaño del alfabeto. Este valida que los valores sean numéricos; en caso de que no sea así, muestra por pantalla el mensaje Çlave inválida, revise que solo se haya ingresado números separados por espacios".

- El método char[][] generaMatrizAlfabeto() genera la matriz del alfabeto, la cual sirve para el cifrado que estamos buscando.
- String encrypt(String message): este método cifra el mensaje carácter por carácter, buscando la fila correspondiente y obteniendo el cifrado de la matriz alphabet.

```
public String encrypt(String message) {  2 usages
    StringBuilder resultado = new StringBuilder();
    for (int i = 0; i < message.length(); i++) {
        char caracterActual = message.charAt(i);
        int fila = character.indexOf(caracterActual);
        if (fila == -1) {
            throw new IllegalArgumentException("El mensaje contiene caracteres no soportados: " + caracterActual); }
        int col = key[i % key.length];
        resultado.append(alphabet[fila][col]); }
    return resultado.toString(); }
```

Figura 3: String Encrypt

- String decrypt(String encryptedMessage): Es el contrario del método encrypt, ya que recorre la columna del carácter cifrado y recupera el original.

```
public String decrypt(String encryptedMessage) {  2 usages
    StringBuilder original = new StringBuilder();
    for (int i = 0; i < encryptedMessage.length(); i++) {
        char caracterActual = encryptedMessage.charAt(i);
        int col = key[i % key.length];
        int fila = -1;

        for (int j = 0; j < alphabet_size; j++) {
            if (alphabet[j][col] == caracterActual) {
                fila = j;
                break;}
        }
        if (fila == -1) {
            throw new IllegalArgumentException("El mensaje contiene caracteres no soportados.");}
        original.append(character.charAt(fila));}
    return original.toString(); }
```

Figura 4: String Decrypt

- Void reEncrypt(): Este permite combinar el cifrado y el descifrado directamente, reencriptando el mensaje con una nueva clave.
- char search (int position) recorre la matriz fila por fila buscando un carácter en especifico.
- char optimalSearch(int position) Este método optimiza la búsqueda, ya que no recorre toda la matriz, sino que accede directamente a la posición.

```
Ejemplo:
Mensaje original: Laboratorio1Ñ
Mensaje cifrado: Ñcgvucyvukt8Q
Mensaje descifrado: Laboratorio1Ñ

(Practica) Para decifrar:
Introduce el mensaje cifrado: Hola123ñ
Mensaje descifrado: Eng3Y0Yh
Nueva clave (numeros separados por espacios): 1 2 3
Clave guardada y cifrada correctamente.
Texto reencriptado: Foj403Zj
```

Figura 5: Void reEncrypt

```
public char optimalSearch(int position) {  no usages
    if (position < 0 || position >= alphabet_size * alphabet_size) {
        throw new IllegalArgumentException("Posicion fuera de rango.");
    }

    int fila = position / alphabet_size;
    int col = position % alphabet_size;
    return alphabet[fila][col];
}
```

Figura 6: Class BigVigenere

4. Marco Teórico

El cifrado Vigenere, es un método que se extiende desde su creación en el siglo XVI y XVII, este método de cifrado polialfabetico posee una estructura que abarca puntos tales como el cifrado y descifrado de claves introducidas al código, en el caso de este laboratorio el equipo se enfoco en utilizar el cifrado BigVigenere (una adaptación moderna del cifrado Vigenere) enfocándose en introducir una clave y transformándola a través de encriptaciones y desencriptaciones, para lograr obtener la clave original, todo esto con el objetivo de lograr aprender a utilizar el cifrado Vigenere

5. Experimentación

A lo largo del laboratorio el equipo se focalizo en realizar experimentos que concluyeron la eficacia del código realizado, interesándose en visualizar la efectividad de la clase BigVigenere y los métodos de encriptacion y descriptacion, logrando obtener datos que futuramente fueron aplicados a la gráfica, a su vez logrando obtener la

notación Big O que demuestra el comportamiento del tiempo de ejecución o el uso de memoria de un algoritmo en función del tamaño de la entrada.

5.1. Ejemplo mensaje a cifrar:

Mensaje original: Laboratorio1Ñ
Mensaje cifrado: ñcgvucyvukt8Q
Mensaje descifrado: Laboratorio1Ñ

5.2. Grafico de tiempo de ejecución:

Largo Key	Tiempo Cifrado (ms)	Tiempo Descifrado (ms)
10	6	23
50	4	4
100	2	7
500	2	4
1000	3	1
5000	3	7

Cuadro 1: Tiempo de cifrado y descifrado según el largo de Key

Figura 7: Grafico de Largo y Tiempo

5.3. Comparación de efciciencia (Notacion Big O)

Ya que utilizamos el metodo optimalSearch logramos que este codigo sea eficiente respecto al tamaño del mensaje, sin embargo, el peor caso en este codigo es cuando el mensaje por decifrar es muy largo y/o la clave es muy larga, esto quiere decir que su complejidad es **denotada como O(m + k)** siendo k la cantidad de numeros que tiene la clave (ejemplo clave: 1 2 3) k=3.

6. Análisis y Resultados

En este laboratorio, el grupo logró identificar la eficacia y notación Big O del código, logrando identificar cómo el tiempo de ejecución crecía en relación con la longitud del mensaje y la clave, tal como se muestra en el siguiente gráfico.

Figura 8: Class BigVigenere

Class BigVigener

En la siguiente imagen se podrá observar el resultado que se obtuvo de este código, lo que se ve por pantalla:

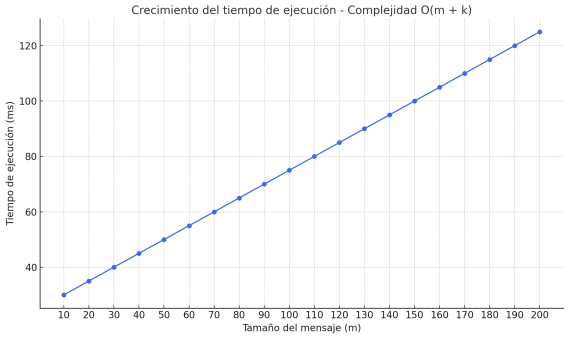

Figura 9: Gráfico

7. Conclusiones

7.1. ¿Cuales dificultades se presentaron y cómo se resolvieron?

A lo largo del laboratorio se presentaron ciertas complicaciones a la hora de agregar el método de OptimalSearch, ya que en un inicio el equipo no comprendía cómo crear un método para acceder directamente a la posición luego de haber utilizado todos los métodos anteriores. Sin embargo, esta problemática fue rápidamente resuelta con la condición ïf", logrando acceder a la posición de manera más limpia y ordenada, tal como lo indicaba el código.

7.2. ¿Cómo se logró optimizar la busqueda de las posiciones?

La busqueda de las posiciones se logro optimizar gracias al metodo OptimalShearch, metodo el cual, como su nombre lo indica, buscaba optimizar la busqueda de una posicion dentro del arreglo, logrando identificarlo gracias a la condicional IF.

7.3. ¿Qué recomendaciones entregaría para la creacion de la clave?

Se recomienda la creación de una clave con pocos caracteres, pero con una mezcla entre mayúsculas, minúsculas y números, además de la posible implementación de la Ñ, ya que con estas combinaciones de caracteres se presenta una clave más segura, pero a su vez más rápida e impecable de encriptar y desencriptar para el uso del usuario.

7.4. Desenlace

Finalmente, se puede determinar que el código realizado fue ejecutado con éxito, logrando demostrar que el equipo logró implementar de manera adecuada y efectiva el cifrado Vigenere, creando una clase BigVigenere en Java, la cual les permitió cifrar y descifrar mensajes de manera eficaz.

Evidenciando a través de gráficos y notaciones la eficacia del código creado, siendo este mismo un código de eficacia lineal (mas que todo al final del grafico, ya que al inicio igual se muestra un desbalance en los datos) demostrandose que al aumentar los caracteres de la clave aumenta a su vez la encriptación y la decriptación del mismo.