

Taller No.1 de Seguridad Computacional

Cristian Andrés Basto Largo

Edwin Camilo Becerra Meche

Gabriel Fernando Castillo Mendieta

Laura Daniela Vargas Jiménez

Facultad de Ingeniería, Universidad Pedagógica y Tecnológica de Colombia.

8108282: Electiva IV: Seguridad Computacional

Ing. Alex Puertas Gonzales

7 de abril del 2025

Tabla De Contenido

Introducción	2
Fundamentaciones y Consideraciones	3
Ethical Considerations in Computational Security	3
Mapa Conceptual Seguro	5
Modelando Accesos a los Datos	5
Modelo Bell LaPadula	6
Modelo Biba.....	8
Cifrando y Descifrando	10
Un Mensaje Cifrado.....	10
Cifrado Afín.....	11
Definición.	11
Descifrado.....	12
Desarrollo y Aplicación	13
Código Demostrativo	15
Cifrando Un Vigenère Con Programación.....	15
Definición	15
Código Demostrativo	17
Un Poco De Contexto.	17
Explicación Código.....	18
Ciframiento de Bloques	22
Proceso De Cifrado	24
Generación De Claves Y Vector De Inicialización (IV)	24
División Del Texto En Bloques	25
Cifrado De Cada Bloque.....	26
Obtención Del Texto Cifrado.....	29
Proceso De Descifrado.....	29
División Del Texto Cifrado En Bloques	29
Descifrado de Cada Bloque (en orden inverso).	30
Reconstrucción del Texto Original.	32
Visualización de resultados.....	32
Cifrado	32
Descifrado.....	34
Diagrama de bloques y repositorio	34
Conclusiones	36
Referencias.....	37

Introducción

La seguridad informática, “se enfoca en minimizar los riesgos y vulnerabilidades en los recursos de hardware y software relacionados con el acceso y la utilización malintencionada de la información de los sistemas de software, para garantizar la integridad, confidencialidad y disponibilidad de esta” (Coronel Suárez & Quirumbay Yagual, 2022).

Un aspecto fundamental en este campo es la evaluación de riesgos y vulnerabilidades, ya que permite detectar posibles puntos débiles antes de que sean explotados. Para esto, se emplean modelos como Bell-LaPadula y Biba, que establecen reglas para controlar el acceso y asegurar la integridad de la información.

Actualmente, la difusión de la información y la comunicación se han trasladado al entorno digital, lo que hace indispensable garantizar la seguridad en línea, el almacenamiento de datos sensibles y la autenticación de usuarios. En este contexto, la criptografía juega un papel esencial, ya que utiliza algoritmos de cifrado que permiten que solo las personas autorizadas accedan a la información.

De igual forma, la ética en la seguridad informática es indispensable. No solo se trata de proteger sistemas, sino de hacerlo de manera responsable, respetando la privacidad y los derechos de los usuarios. La divulgación de vulnerabilidades, las pruebas de seguridad y el manejo de la información deben seguir principios éticos para evitar daños innecesarios.

Por último, organizar y visualizar estos conceptos de forma clara, como en mapas conceptuales, facilita su comprensión. Estas herramientas ayudan a conectar ideas y entender mejor cómo funciona la seguridad en entornos tecnológicos, permitiendo una visión más estructurada y accesible del tema.

Fundamentaciones y Consideraciones

Ethical Considerations in Computational Security

Ethical considerations in computer security Computer security has many implications, one of which is ethics. Ethics is responsible for protecting systems and user data in the modern digital landscape. Due to the growing demand for technology and its rapid advancement, ethics must go beyond technical competence; therefore, professionals must assess the social, organisational and individual implications of their actions.

What are the ethical implications of computer security audits?

Security assessments are essential for identifying and mitigating system vulnerabilities. According to the MKS (2023), "obtaining formal permissions and adhering to ethical codes of conduct form the basis of responsible security practices". Ethical hacking, when conducted within a legal and authorised framework, provides critical insights by simulating real-world attack scenarios. However, the line between ethical behaviour and misconduct can be thin. Conducting tests without prior consent or misusing discovered vulnerabilities can result in both legal and ethical violations, affecting not only the targeted individual or organisation, but also the attacker or researcher responsible.

The discovery of a security vulnerability often places researchers in ethically complex situations. One such dilemma is determining the appropriate timing and method of disclosure. Responsible disclosure involves notifying the affected party in confidence and allowing time for remediation before public disclosure. Conversely, full disclosure involves immediate public notification, which may put users at risk if the vulnerability is exploited before a fix is in place. As noted in Ethics, the Human Factor (2016), "striking a balance between transparency and

caution is essential: early disclosure may put users at risk, while delayed disclosure may postpone necessary security measures".

Information security professionals often rely on ethical frameworks to guide their decision-making. Silva and Espina (2006) argue that "these models help professionals navigate complex scenarios, such as deciding whether to cooperate with law enforcement agencies in surveillance programs or to disclose vulnerabilities that could damage an organisation's reputation".

Ethical dilemmas in cybersecurity often manifest themselves in practical situations. A clear example is hacking competitions, where participants may discover valuable zero-day vulnerabilities. Although ethical practice dictates that these findings be reported through responsible channels, the temptation to sell them to third parties remains. Such actions can facilitate malicious exploitation, putting countless users at risk. In addition, MKS (2023) notes that "government agencies may withhold vulnerability disclosures for surveillance purposes". While such actions may support national security, they also risk violating individual privacy and undermining public trust in institutions.

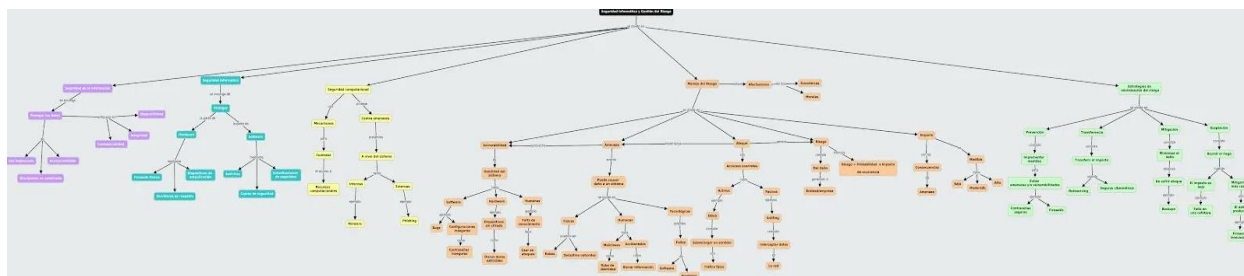
In conclusion, ethics should be seen as an essential part of information security, not just an optional consideration. Professionals are charged not only with protecting systems and data, but also with defending the rights of individuals and the well-being of society as a whole. As Silva and Espina (2006) state, "Integrity must guide all aspects of information security, from system assessment to vulnerability disclosure and policy development". As threats continue to evolve, so must the ethical standards that govern the profession to ensure a secure and principled digital future.

Mapa Conceptual Seguro

La figura 1 presenta un mapa conceptual que integra definiciones fundamentales en seguridad computacional, incluyendo seguridad de la información, informática y computacional; junto con los componentes esenciales de la gestión del riesgo: amenaza, ataque, vulnerabilidad y riesgo. Además, se sintetizan las principales estrategias de tratamiento del riesgo: prevención, transferencia, mitigación y aceptación. Para una mejor visualización, revisar el apéndice A.

Figura 1

Seguridad informática y gestión del riesgo

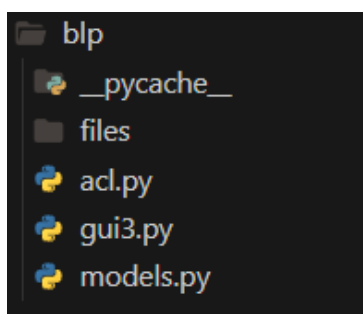


Modelando Accesos a los Datos

Se utilizó Python para simular los modelos Bell-LaPadula (BLP) y Biba mediante listas de control de acceso (ACLs). El código sigue los principios de: BLP: "No read up, no write down" y Biba: "No read down, no write up". Para una mejor comprensión revisar apéndice B.

Figura 2

Estructura del proyecto de Bell LaPadula

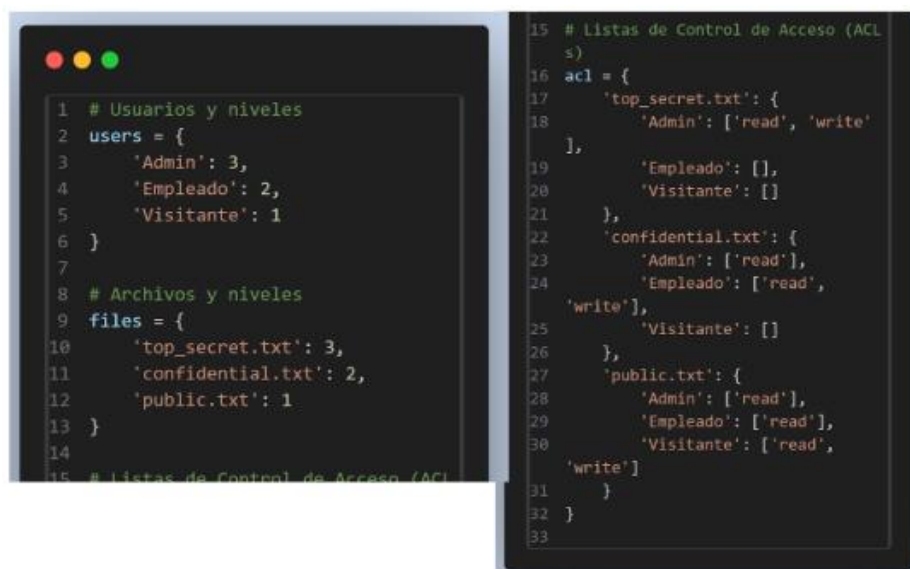


Modelo Bell LaPadula

En la Figura 2 el archivo “acl.py” se especifican las acciones que rigen el modelo, asignando el peso de los usuarios y de los textos.

Figura 3

Clase “acl.py” del proyecto de Bell LaPadula



```
1 # Usuarios y niveles
2 users = {
3     'Admin': 3,
4     'Empleado': 2,
5     'Visitante': 1
6 }
7
8 # Archivos y niveles
9 files = {
10     'top_secret.txt': 3,
11     'confidential.txt': 2,
12     'public.txt': 1
13 }
14
15 # Listas de Control de Acceso (ACL
16 s)
17 acl = {
18     'top_secret.txt': {
19         'Admin': ['read', 'write'
20 ],
21         'Empleado': [],
22         'Visitante': []
23     },
24     'confidential.txt': {
25         'Admin': ['read'],
26         'Empleado': ['read',
27 'write'],
28         'Visitante': []
29     },
30     'public.txt': {
31         'Admin': ['read'],
32         'Empleado': ['read'],
33         'Visitante': ['read',
34 'write']
35     }
36 }
```

En la figura 3 se puede apreciar, el comportamiento que se le brinda al modelo regido por los niveles y usuarios, además de sus acciones

Tabla 1

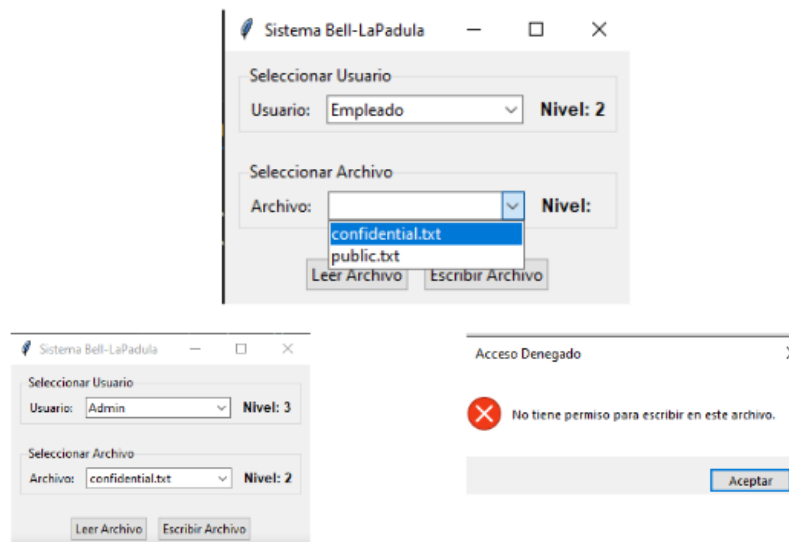
Matriz de acceso del modelo Bell LaPadula.

Users	Top_Secret (3)	Confidential (2)	Public (1)
Admin (3)	Leer/Escribir	Leer	Leer
Empleado (2)	Sin Acceso	Leer/Escribir	Leer
Visitante (1)	Sin acceso	Sin Acceso	Leer/Escribir

En la tabla 1, se puede apreciar de manera más clara, el comportamiento mencionado anteriormente, donde se especifican las acciones correspondientes a los usuarios a continuación, se observa un caso de ejecución.

Figura 4

Salida esperada para el caso de: nivel 2 para acceder a un archivo nivel 3.



En la figura 4 se puede observar que el usuario nivel dos no tiene conocimiento de los archivos de mayor nivel, por ende, aseguramos confidencialidad en ese aspecto, además, si tratamos de escribir un archivo de menor nivel, no se estará permitido.

Modelo Biba

Figura 5

Estructura del proyecto del modelo Biba

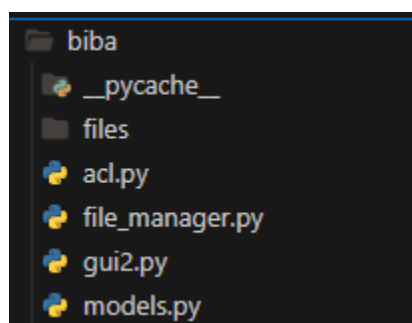


Figura 6

Clase "acl.py" del proyecto de Biba



En la figura 6 del archivo acl.py se especifican las acciones que rigen al modelo.

Tabla 2

Matriz de acceso para el modelo Biba

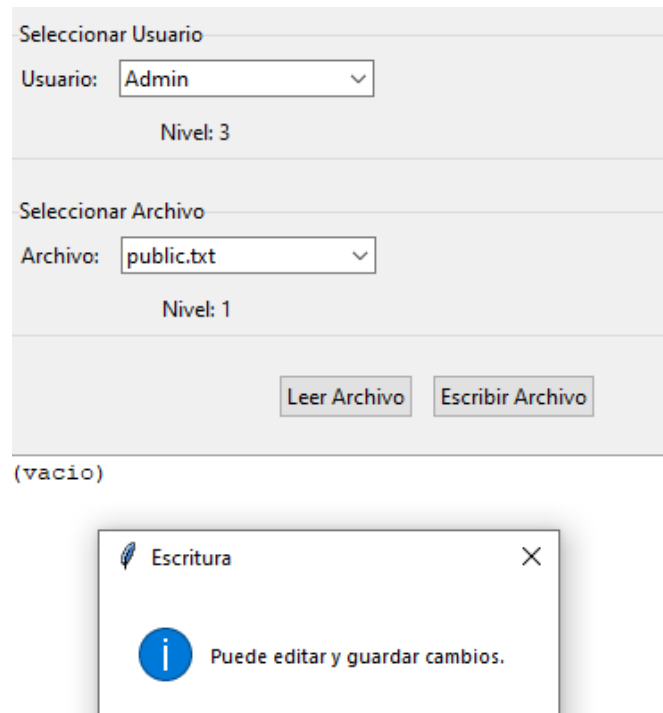
Users	Top_Secret (3)	Confidential (2)	Public (1)
Admin (3)	Leer/Escribir	Escribir	Escribir
Empleado (2)	Leer	Leer/Escribir	Escribir
Visitante (1)	Leer	Leer	Leer/Escribir

En la tabla 2 se puede apreciar la matriz de acceso del modelo Biba que describe de manera más legible, las reglas por las cuales se simuló el modelo Biba.

Un ejemplo de ejecución, donde un usuario de mayor nivel puede escribir en un archivo de menor nivel, la salida esperada sería correcta.

Figura 7

Salida exitosa para el caso de: usuario de nivel superior quiera escribir en un archivo de nivel inferior



The screenshot shows a web application interface with two main sections. The first section, titled 'Seleccionar Usuario', contains a dropdown menu for 'Usuario' with 'Admin' selected, and a label 'Nivel: 3'. The second section, titled 'Seleccionar Archivo', contains a dropdown menu for 'Archivo' with 'public.txt' selected, and a label 'Nivel: 1'. Below these sections are two buttons: 'Leer Archivo' and 'Escribir Archivo'. At the bottom of the interface, the text '(vacio)' is displayed. A small dialog box titled 'Escritura' is open, showing an information icon and the text 'Puede editar y guardar cambios.'

Cifrando y Descifrando

Un Mensaje Cifrado

Se ha podido capturar este mensaje en una transmisión entre dos puntos. No fue posible obtener K_c , pero se presume que la cadena es descifrable, Se debe obtener el texto claro a partir del siguiente ciframiento:

AEUAMXSIJIRIVBMRIVJMJIZBYUYDYUJIBYTVYUYBMAIZTIBMXIVUYZMWSIFM
A EVMJYFMKEMJIZTVYJIUE

Para esto, se ha hecho uso del tipo de desciframiento Afín, el cual es un cifrado por sustitución y a su vez se realizó una transposición manual para la coherencia del mensaje.

Para nuestro desarrollo se automatizó el descifrador en una hoja de Excel y se hizo un breve programa en Python para demostrar su funcionamiento de una forma visual e interactiva. Dicho código solo es una extensión de lo desarrollado en el documento Excel, por lo cual, en el código se implementa directamente el algoritmo y para el ordenamiento de las palabras se sabe de antemano cuál es el mensaje descifrado.

Cifrado Afín

Definición.

El cifrado afín es un tipo de cifrado por sustitución en el que cada letra del mensaje original es transformada mediante una función matemática. Es un método clásico de encriptación basado en la aritmética modular y se considera una mejora del cifrado César, ya que introduce dos operaciones en lugar de una.

Para cifrar un mensaje con el método afín, cada letra del alfabeto se convierte en un número utilizando una tabla de equivalencia. En el alfabeto inglés de 26 letras, entonces, De la A a la Z estaría enumerado desde el 0 al 25.

Una vez convertida cada letra en su equivalente numérico, se aplica la siguiente función matemática de cifrado de la forma:

$$E(x) = (ax+b) \bmod m,$$

Donde:

- x es el número correspondiente a la letra original.
- a y b son claves enteras que definen la transformación.
- m es el tamaño del alfabeto (en este caso, $m = 26$).
- a debe ser coprimo con m para garantizar que el cifrado pueda revertirse.

Dos números son coprimos (o primos entre sí) si el único número que los divide a ambos es 1. Por ejemplo:

- 9 y 26 son coprimos porque su único divisor común es 1.
- 10 y 26 no son coprimos porque ambos son divisibles por 2.

En el cifrado afín, es necesario que a sea coprimo con m para que exista un inverso multiplicativo que permita descifrar el mensaje. Los valores de a posibles para un alfabeto de 26 letras son: 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23 y 25.

Descifrado.

Para recuperar el mensaje original, se usa la función inversa:

$$D(y) = a^{-1} * (y - b) \bmod m,$$

donde:

- y es el número de la letra cifrada.
- a^{-1} es el inverso multiplicativo de a módulo m .
- b es la misma constante usada en el cifrado.

Para calcular a^{-1} , se usa el algoritmo extendido de Euclides, que encuentra un número a^{-1} tal que:

$$(a \cdot a^{-1}) \bmod m = 1$$

Si $a = 5$ en un alfabeto de 26 letras, su inverso multiplicativo es 21, porque:

$$(5 \times 21) \bmod 26 = 105 \bmod 26 = 1$$

Con este valor, el mensaje puede descifrarse correctamente.

El cifrado afín es un método de encriptación sencillo basado en operaciones matemáticas básicas. Su seguridad depende de la elección correcta de los valores a y b , y de que a sea

coprimo con m para garantizar la reversibilidad del cifrado. Aunque es fácil de entender y aplicar, su seguridad es limitada, ya que puede ser vulnerado mediante análisis de frecuencia o ataques de fuerza bruta.

Desarrollo y Aplicación

Inicialmente se comenzó barajando qué métodos de desciframiento podría aplicarse al mensaje cifrado. Primero se probó con César en una calculadora online, pero al dar todo el recorrido no daba ni un solo resultado. A falta de una llave o clave el tipo de cifrado Vigenère fue descartado inmediatamente.

Al hacer investigación se llegó a que el mensaje era posiblemente un cifrado tipo Afín. Con esta información y después de una investigación se procedió a crear un documento en Excel para automatizar este método aplicado exclusivamente para descifrar mensajes cifrados por afín. En el apéndice 3 se presenta la plantilla de Excel para el descifrador afín.

Para ello, los valores de Mod, Decimación (a) y Desplazamiento (b) se pueden modificar según quiera el usuario. La casilla a^{-1} hace directamente el inverso del valor de decimación ingresado. En el documento se evidencia el mensaje cifrado detectado y debajo el carácter descifrado según los valores ingresados en a y b . (Ver figura 8)

Figura 8

Datos usados en el “Descifrador Afín”

Mod	26	
Decimación (a)	25	
Desplazamiento (b)	12	
a^{-1}	25	
AFIN DECIPHER	$P = a^{-1} * (C - b) \text{ Mod } 26$	

Figura 9

Descripción de los colores para los caracteres del texto cifrado y descifrado

P. DESCIFRADO
P. CIFRADO

En el documento aparecen otras tablas relacionadas al desciframiento afín. Con una metodología de prueba y error se llegó a que los valores que mostraron un mensaje legible fueron para $a = 25$ y para $b = 12$. Dando como resultado el siguiente mensaje:

MISMA PUEDE VER LA VERDAD EN LOS OJOS DEL OTRO SOLAMENTE LA
PERSONA QUE HA MIRADO HACIA DENTRO DE SI

El mensaje no tiene sentido si se lee tal cual, lo que da a suponer una transcripción del mismo. A observación se ve que el mensaje se divide en dos bloques, el primero siendo: ‘MISMA PUEDE VER LA VERDAD EN LOS OJOS DEL OTRO’ y el segundo ‘SOLAMENTE LA PERSONA QUE HA MIRADO HACIA DENTRO DE SI’.

Entonces si pasamos el primer segundo bloque a la primera posición nos queda:
“SOLAMENTE LA PERSONA QUE HA MIRADO HACIA DENTRO DE SÍ MISMA PUEDE
VER LA VERDAD EN LOS OJOS DEL OTRO”.

Como resultado obtenemos un mensaje completamente legible y coherente.

Código Demostrativo

En el código ya se sabía el resultado obtenido con anterioridad gracias a lo trabajado en la hoja de cálculo presentada anteriormente, es por eso que el mensaje organizado y coherente aparece directamente en el código de programa como una validación en la función *reordenar_frase*, al usar otros valores solo genera el texto “descifrado” pero sin añadir un orden, solo deja el texto igual, el código se puede encontrar en el apéndice D.

Cifrando Un Vigenère Con Programación

Definición

El Cifrado de Vigenère es un método de cifrado de sustitución poli-alfabético que utiliza una clave repetitiva para transformar un texto en claro en un texto cifrado. Este cifrado se basa en la suma modular de los valores numéricos de los caracteres del mensaje original y los de la clave, garantizando una mayor seguridad frente a los ataques de frecuencia utilizados en cifrados mono-alfabéticos como el cifrado César.

El cifrado opera siguiendo la siguiente ecuación matemática:

$$Y_i = (X_i + Z_i) \bmod T$$

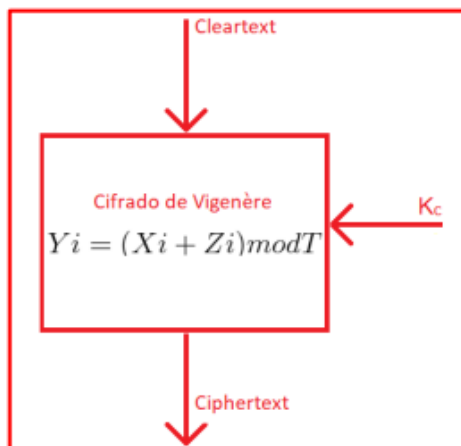
Donde:

- X_i representa el carácter i -ésimo del texto en claro, convertido a un valor numérico.
- Z_i es el carácter correspondiente de la clave, también convertido en valor numérico.
- T es el tamaño del alfabeto utilizado (por ejemplo, 26 para el alfabeto inglés).
- Y_i es el carácter cifrado resultante tras aplicar la operación.

Este proceso se realiza para cada carácter del mensaje, repitiendo la clave en caso de que su longitud sea menor que la del texto en claro.

Figura 10

Algoritmo de cifrado Vigenère base



Una de las principales ventajas del Cifrado de Vigenère es que proporciona una mayor seguridad en comparación con los cifrados mono-alfabéticos, como el Cifrado César. Al utilizar una clave repetitiva, este método introduce variabilidad en el cifrado de cada letra, lo que dificulta los análisis de frecuencia utilizados para romper cifrados más simples. Además, es un algoritmo fácil de implementar y comprender, lo que lo hace accesible tanto para aprendizaje como para aplicaciones prácticas en contextos donde no se requiere una seguridad extremadamente alta. También es útil para cifrar mensajes cortos cuando se dispone de una clave secreta conocida sólo por el emisor y el receptor.

A pesar de sus ventajas, el Cifrado de Vigenère presenta vulnerabilidades significativas. Si la clave utilizada es demasiado corta o predecible, el cifrado puede ser fácilmente descifrado mediante ataques como el criptoanálisis de Kasiski o el análisis de coincidencias de Friedman. Además, aunque este método era considerado irrompible durante siglos, hoy en día se puede descifrar con técnicas computacionales avanzadas, lo que lo hace inadecuado para la protección de información sensible en entornos modernos. Otra desventaja es que requiere que el emisor y

el receptor acuerden y mantengan en secreto la clave, lo que puede representar un problema logístico en algunos escenarios.

El Cifrado de Vigenère fue considerado irrompible durante siglos, hasta que se descubrieron métodos para descifrarlo sin conocer la clave. Hoy en día, aunque ya no es utilizado para proteger información sensible, sigue siendo un concepto fundamental en la criptografía clásica y un punto de partida para el desarrollo de cifrados más avanzados.

Código Demostrativo

Un Poco De Contexto.

Se requiere la sistematización del algoritmo de ciframiento de Vigenère.

Para esto, la idea es que se desarrollé un programa en algún lenguaje respectivo, que pueda desarrollar ese algoritmo de cifrado, teniendo en cuenta las siguientes características:

- Se debe poder ingresar, tanto la clave para el cifrado del mensaje, como el texto claro, a través de la interfaz de usuario.
- No se implementará desde el algoritmo, alfabetos que incluyan la letra ñ ni el carácter espacio.
- Debe mostrarse por pantalla la cadena cifrada producto de la implementación del algoritmo.

En resumen, se requiere un programa que cifre un mensaje con base a una clave también ingresada por el usuario y una fórmula perteneciente al cifrado Vigenère. El cifrado es de una sola vía, solo se ingresa el mensaje claro y retorna el mensaje cifrado.

Explicación Código

El programa fue realizado en Python implementando el cifrado Vigenère según su fórmula. Se realizó una interfaz gráfica sencilla donde se ingresa tanto el mensaje claro como la clave de ciframiento, al final el botón de Cifrar y como resultado el mensaje cifrado. (Ver figura 11)

Figura 11

Interfaz del programa del cifrador Vigenère



En la figura 12 se presenta el ejemplo del mensaje a cifrar, su clave y el mensaje completamente cifrado con las validaciones expuestas anteriormente (no espacios ni caracteres especiales no alfabéticos como la ñ):

Figura 12

Ejemplo del cifrado usando Vigenère



The image shows a web application window titled "Cifrado de Vigenère". It has a dark blue background. At the top, the title "Cifrado de Vigenère" is displayed in white. Below the title, there are two input fields. The first is labeled "Texto Claro:" and contains the text "basto becerra castillo vargas". The second is labeled "Clave:" and contains the text "simulación". Below these fields is a red button with the text "Cifrar" in white. At the bottom of the window, the result is displayed in white text: "Texto cifrado: TIENZBGKSEJIOUDTKTZBNIDALS".

Para lograr el cometido fue necesaria una función que leyera la cadena completa y le hiciera una “limpieza” para eliminar esos valores.

La siguiente función prepara el texto para el cifrado. Convierte todas las letras a mayúsculas, reemplaza caracteres con tilde (como á, é, í...) por su versión sin tilde, y cambia la "ñ" por "N". Además, elimina cualquier símbolo, número o espacio, dejando solo letras del alfabeto con ayuda del método *isaplha* ().

Figura 13

Función para limpiar los datos

```
def limpiar_texto(texto):  
    REEMPLAZOS = {  
        'á': 'A',  
        'é': 'E',  
        'í': 'I',  
        'ó': 'O',  
        'ú': 'U',  
        'ñ': 'N', # La ñ se convierte en N mayúscula  
    }  
  
    texto_limpio = []  
    for c in texto:  
        REEMPLAZOS.get(c.lower(), c.upper()) # Reemplaza tildes y ñ  
        if c.isalpha() # Solo permite letras del alfabeto  
    return ''.join(texto_limpio)
```

En esta función se adapta la clave al tamaño del texto que se va a cifrar. Primero, limpia la clave (quitando tildes y dejando solo letras mayúsculas). Luego, la repite las veces necesarias para que tenga al menos el mismo largo que el texto. Finalmente, la recorta si se pasa. Esto es necesario porque en el cifrado Vigenère cada letra del texto se cifra usando una letra de la clave en la misma posición.

Figura 14

Función para generar claves

```
def generar_clave(texto, clave):  
    clave = limpiar_texto(clave)  
    return (clave * (len(texto) // len(clave) + 1))[:len(texto)]
```

Ahora con eso listo se aplica el cifrado Vigenère al texto. Lo primero es limpiar el texto y luego ajustar la clave. Se define un alfabeto estándar inglés de A a Z para usar como base del

cifrado. La parte más importante está en el bucle for, donde se usa zip (texto, clave) para emparejar letra por letra del texto con su correspondiente en la clave.

Luego, para cada par de letras (t del texto y k de la clave), se buscan sus posiciones en el alfabeto y se suman. El resultado se ajusta con el operador módulo (% 26) para mantenerse dentro del alfabeto. Esto implementa la fórmula del cifrado: $Y_i = (X_i + Z_i) \bmod T$.

La letra resultante de esa suma es la que se agrega al texto cifrado. Al final, se devuelve todo el mensaje cifrado completo.

Figura 15

Función para cifrar por el método Vigenère

```
def cifrar_vigenere(texto, clave):  
    texto = limpiar_texto(texto) # Se limpia el texto  
    clave = generar_clave(texto, clave) # Se ajusta la clave a la longitud del texto  
    alfabeto = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'  
  
    texto_cifrado = ''  
    for t, k in zip(texto, clave):  
        # Se calcula el nuevo índice sumando los valores del texto y la clave  
        nuevo_indice = (alfabeto.index(t) + alfabeto.index(k)) % len(alfabeto)  
        texto_cifrado += alfabeto[nuevo_indice]  
  
    return texto_cifrado
```

Esta función recoge el texto y la clave desde la interfaz gráfica. Si alguno de los dos campos está vacío, muestra un mensaje de error. Si ambos están completos, aplica el cifrado Vigenère usando la función principal y muestra el resultado en pantalla. Es la función que conecta lo que el usuario escribe con el proceso de cifrado.

Figura 16

Función de entrada de los datos por parte del usuario

```
def cifrar():  
    texto_claro = entrada_texto.get()  
    clave = entrada_clave.get()  
    if not texto_claro or not clave:  
        messagebox.showerror("Error", "Debe ingresar texto y clave.")  
        return  
    texto_cifrado = cifrar_vigenere(texto_claro, clave)  
    resultado_label.config(text=f"Texto cifrado: {texto_cifrado}")
```

El repositorio para su observación y ejecución del programa está ubicado en el apéndice

E.

Ciframiento de Bloques

El cifrador de bloques desarrollado utiliza el framework Express para su implementación.

La lógica central del algoritmo reside en la clase BlockCipher como se observa en la Figura 19,

donde se definen los parámetros principales, tales como:

- Número de rondas: 32 rondas de cifrado y descifrado.
- Tamaño del bloque: Cada bloque consta de 4 caracteres.
- Alfabeto: Se usa el alfabeto en inglés (a-z), omitiendo la letra ñ.
- Clave secreta: Puede ser proporcionada por el usuario o generada por defecto.

Figura 17

Definición de parámetros usados



```
constructor(secretKey, iv = null) {  
  this.numRounds = 32;  
  this.blockSize = 4;  
  this.alphabet = 'abcdefghijklmnopqrstuvwxyz';  
  this.defaultKey = 'defaultsecretkey';  
  this.secretKey = secretKey || this.defaultKey;  
}
```

Esta clave secreta es fundamental, ya que a partir de ella se generan tanto las claves de ronda como el vector de inicialización (IV), los cuales son utilizados en cada fase del proceso de cifrado y descifrado.

Adicionalmente, el algoritmo emplea una clase auxiliar basada en congruencia lineal (LCG, Linear Congruential Generator) para la generación de números pseudoaleatorios. Estos números son clave para la creación del IV y las claves de ronda, además de facilitar la normalización de valores en un rango específico, como se ve entre líneas dentro de la Figura 18.

Figura 18

Función para generar números pseudoaleatorios

```
static calculateTable(x0, a, c, m, iterations, minV  
alue, maxValue) {  
  const table = [];  
  let xi = x0;  
  const niArray = [];  
  
  for (let i = 0; i < iterations; i++) {  
    xi = this.calculateXi(xi, a, c, m);  
    const ri = this.calculateRi(xi, m);  
    const ni = this.calculateNi(minValue, maxValu  
e, ri);  
    niArray.push(ni);  
    table.push([i + 1, xi, ri, ni]);  
  }  
  
  return { table, niArray };  
}
```

Proceso De Cifrado

El proceso de cifrado se desarrolla en varias fases:

Generación De Claves Y Vector De Inicialización (IV)

Se calcula una semilla numérica a partir de la clave secreta, sumando los valores ASCII de cada uno de sus caracteres. Posteriormente, se utiliza un generador de números pseudoaleatorios basado en la ecuación de congruencia lineal:

$$X_{n+1} = (a X_n + c) \bmod m$$

Con los siguientes valores:

- $a = 1664525$
- $c = 1013904223$
- $m = 2^{32}$

A partir de esta secuencia de números pseudoaleatorios se extraen los primeros valores para formar el vector de inicialización (IV), con índices normalizados en el rango [0,25]. Además, se generan 32 conjuntos de claves de ronda, cada uno compuesto por 8 valores representado en la figura 19.

Figura 19

Función para generar llaves

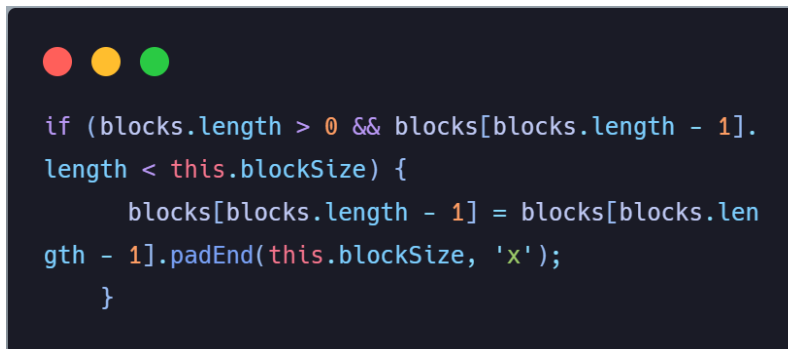
```
generateKeys(secretKey = this.secretKey) {  
  let seed = 0;  
  for (let i = 0; i < secretKey.length; i++) {  
    seed += secretKey.charCodeAt(i);  
  }  
  
  const a = 1664525;  
  const c = 1013904223;  
  const m = Math.pow(2, 32);  
  const iterations = this.numRounds * 8 + this.blockSize;  
  
  const { niArray } = this.calculateLCGTable(seed, a, c, m, iterations, 0, 1);  
  
  const iv = niArray.slice(0, this.blockSize).map((val) => Math.floor(val * 26));  
  const keys = [];  
  for (let i = 0; i < this.numRounds; i++) {  
    const key = niArray.slice(this.blockSize + i * 8, this.blockSize + (i + 1) * 8);  
    keys.push(key);  
  }  
  
  return { keys, iv, lcgParams: { seed, a, c, m } };  
}
```

División Del Texto En Bloques

El mensaje a cifrar se convierte a minúsculas y se eliminan caracteres especiales. En este caso, se divide en bloques de 4 caracteres. Si el último bloque es más corto, se rellena con caracteres de relleno (x) como se ve en las últimas líneas de la Figura 20.

Figura 20

Función para limpiar y dividir el mensaje inicial



```
if (blocks.length > 0 && blocks[blocks.length - 1].  
length < this.blockSize) {  
    blocks[blocks.length - 1] = blocks[blocks.len  
gth - 1].padEnd(this.blockSize, 'x');  
}
```

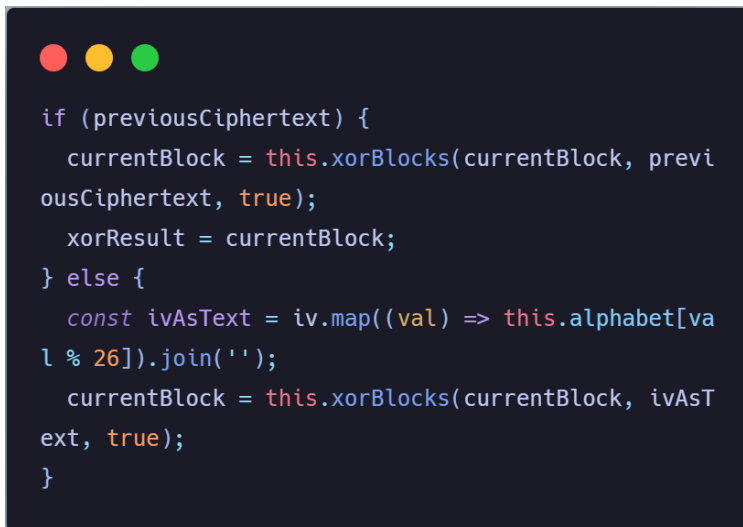
Cifrado De Cada Bloque

Para cada bloque se aplican las siguientes operaciones durante las 32 rondas, de la siguiente manera:

- Operación XOR con el IV o el bloque previo: Si es el primer bloque, se realiza una operación XOR con el vector de inicialización (IV). Si no, se realiza una XOR con el bloque cifrado anterior. Esta se apoya de la función xorBlocks, la cual implementa un cifrado por sustitución tipo XOR sobre caracteres alfabéticos como se observa en la Figura 21.

Figura 21

Operación XOR usando VI o bloque previo



```
if (previousCiphertext) {  
    currentBlock = this.xorBlocks(currentBlock, previ  
ousCiphertext, true);  
    xorResult = currentBlock;  
} else {  
    const ivAsText = iv.map((val) => this.alphabet[va  
l % 26]).join('');  
    currentBlock = this.xorBlocks(currentBlock, ivAsT  
ext, true);  
}
```

- Sustitución de caracteres: Se toma cada carácter del bloque y se convierte a su índice en el alfabeto (a=0, b=1, ..., z=25). Se calcula un nuevo índice con la fórmula: $(\text{índice} + \text{desplazamiento} + IV + \text{ronda}) \bmod 26$ presente en la Figura 24. Se reemplaza el carácter por su nueva posición en el alfabeto como se ve en la Figura 22.

Figura 22

Función para la sustitución de caracteres

```
substitute(block, key, round, iv) {  
  let result = '';  
  const operations = [];  
  for (let i = 0; i < block.length; i++) {  
    const char = block[i];  
    const index = this.alphabet.indexOf(char.toLowerCase());  
    if (index !== -1) {  
      const shift = Math.floor(key[i % key.length] * 25) + 1;  
      const ivShift = iv[i % iv.length];  
      const newIndex = (index + shift + ivShift + round) % 26;  
  
      operations.push(`(${index} + ${shift} + ${ivShift} + ${round}) % 26 = ${newIndex}`);  
      result += this.alphabet[newIndex];  
    } else {  
      operations.push(`${char} (No modificado)`);  
      result += char;  
    }  
  }  
  return { result, operations };  
}
```

- Transposición del bloque: Se invierte el orden de los caracteres para aumentar la complejidad del cifrado como se observa en la figura 23.

Figura 23

Función para la transposición del bloque

```
transpose(block) {  
  return block.split('').reverse().join('');  
}
```

Obtención Del Texto Cifrado

Tras completar las 32 rondas, el bloque cifrado se almacena. En ese sentido, todos los bloques cifrados se concatenan para formar el mensaje final cifrado.

Proceso De Descifrado

El descifrado sigue los mismos pasos en orden inverso:

División Del Texto Cifrado En Bloques

Se separa el texto cifrado en bloques de 4 caracteres como se observa en la Figura 24.

Figura 24

Operación para la división del texto cifrado en bloques

```
const blocks = [];  
for (let i = 0; i < cipherText.length; i += this.blockSize) {  
  blocks.push(cipherText.substring(i, i + this.blockSize));  
}
```

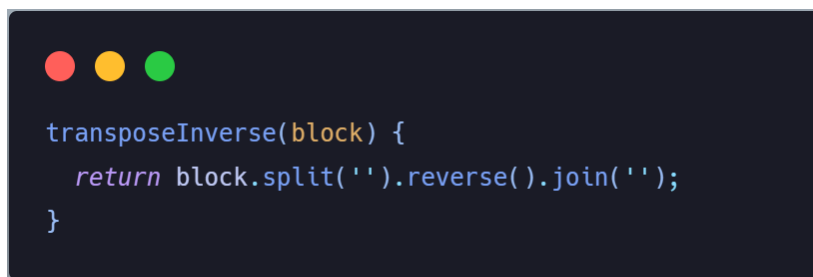
Descifrado de Cada Bloque (en orden inverso).

Para cada bloque se aplican las siguientes operaciones durante las 32 rondas, de la siguiente manera:

- Transposición inversa: Se revierte el orden de los caracteres como se observa en la figura 25.

Figura 25

Operación de transposición inversa



```
transposeInverse(block) {  
  return block.split('').reverse().join('');  
}
```

- Sustitución inversa: Se aplica la fórmula inversa para calcular el índice original, formula presente en la Figura 26:

$$(\text{índice} - \text{desplazamiento} - IV - \text{ronda}) \bmod 26$$

Figura 26*Operación de sustitución inversa*

```
substituteInverse(block, key, round, iv) {
  let result = '';
  const operations = [];
  for (let i = 0; i < block.length; i++) {
    const char = block[i];
    const index = this.alphabet.indexOf(char.toLowerCase());

    if (index !== -1) {
      const shift = Math.floor(key[i % key.length] * 25) + 1;
      const ivShift = iv[i % iv.length];
      let newIndex = (index - shift - ivShift - round) % 26;
      while (newIndex < 0) {
        newIndex += 26;
      }

      operations.push(
        `((${index} - ${shift} - ${ivShift} - ${round}) % 26 + 26) % 26`
      );
      result += this.alphabet[newIndex];
    } else {
      operations.push(`${char} (No modificado)`);
      result += char;
    }
  }
  return { result, operations };
}
```

- Operación XOR inversa: Se aplica XOR con el bloque previo en la secuencia cifrada.

Posteriormente, para el primer bloque, se usa el vector de inicialización (IV) como se ve en la Figura 27.

Figura 27*Operación XOR inversa*

```
if (previousCiphertext) {  
    currentBlock = this.xorBlocks(currentBlock, previousCiphertext, false);  
} else {  
    const ivAsText = iv.map((val) => this.alphabet[val % 26]).join('');  
    currentBlock = this.xorBlocks(currentBlock, ivAsText, false);  
}
```

Reconstrucción del Texto Original.

Se eliminan caracteres de relleno (x), si los hubiera. Posteriormente, se obtiene el mensaje descifrado.

Figura 28*Eliminación de caracteres de relleno*

```
let decryptedText = decryptedBlocks.join('');  
decryptedText = decryptedText.replace(/x+$/, '');
```

Visualización de resultados

La interfaz permite al usuario ingresar una clave opcional (si no la proporciona, se usa la clave por defecto) y el texto a cifrar o descifrar.

Cifrado

En la vista de resultados del cifrado, el usuario encontrará:

- Texto original y texto cifrado: Se muestra el mensaje ingresado y su versión cifrada.
- Configuración del cifrado: Incluye el número de bloques, rondas, longitud de bloque, clave utilizada y el vector de inicialización (IV).

Figura 29

Texto original, cifrado y datos usados

Contenido

TEXTO ORIGINAL

hola

TEXTO CIFRADO

jmpf

Configuración

PARÁMETROS BÁSICOS

Bloques

1

Rondas

32

Longitud de bloque

4

SEGURIDAD

IV

23, 24, 20, 4

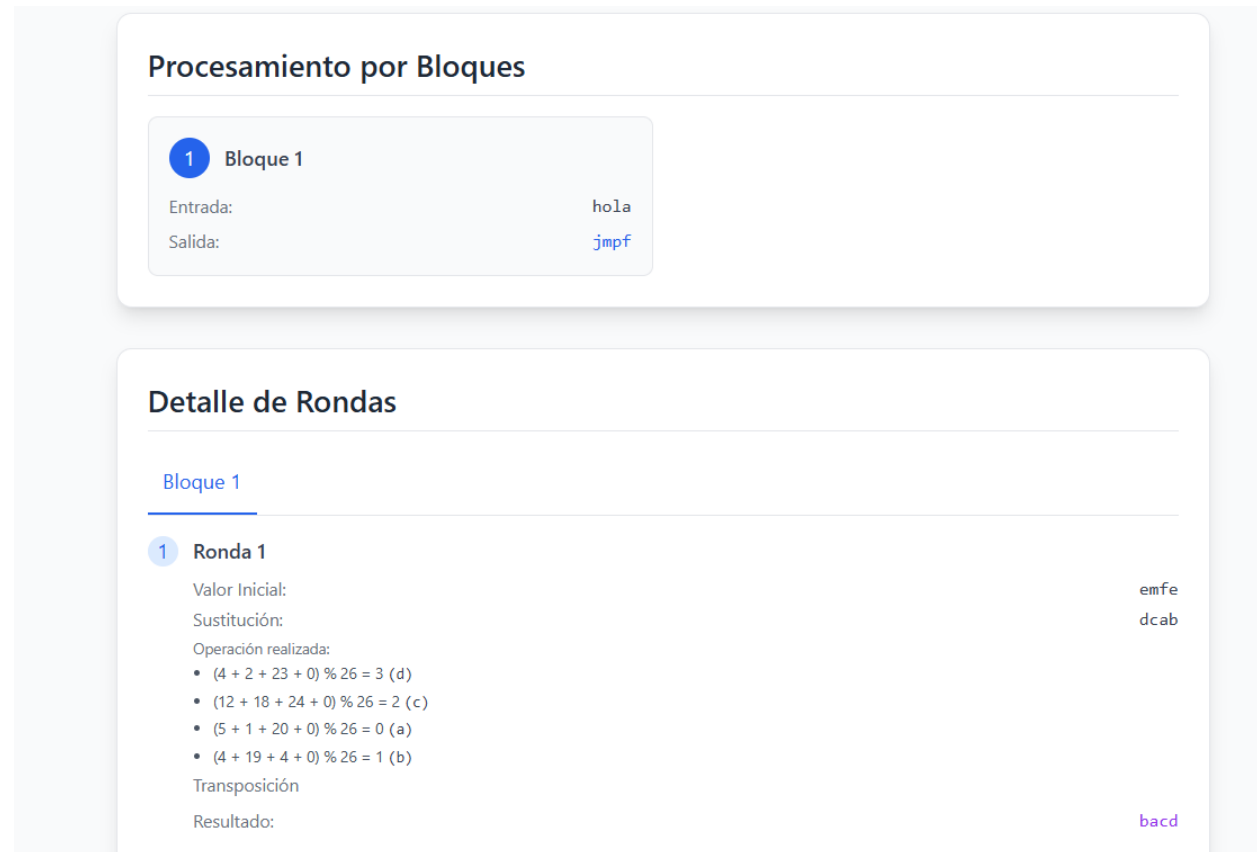
Clave

defaultsecretkey

- Generador de números pseudoaleatorios: Se muestra una breve muestra de los valores generados.
- Procesamiento por bloques: Se presentan hasta 5 bloques del proceso, dado que el sistema admite un máximo de 20 caracteres.
- Evolución por rondas: Se visualiza cómo cambia el texto en cada ronda con la sustitución y transposición, hasta obtener el resultado final.

Figura 30

Resultados de las rondas



Descifrado.

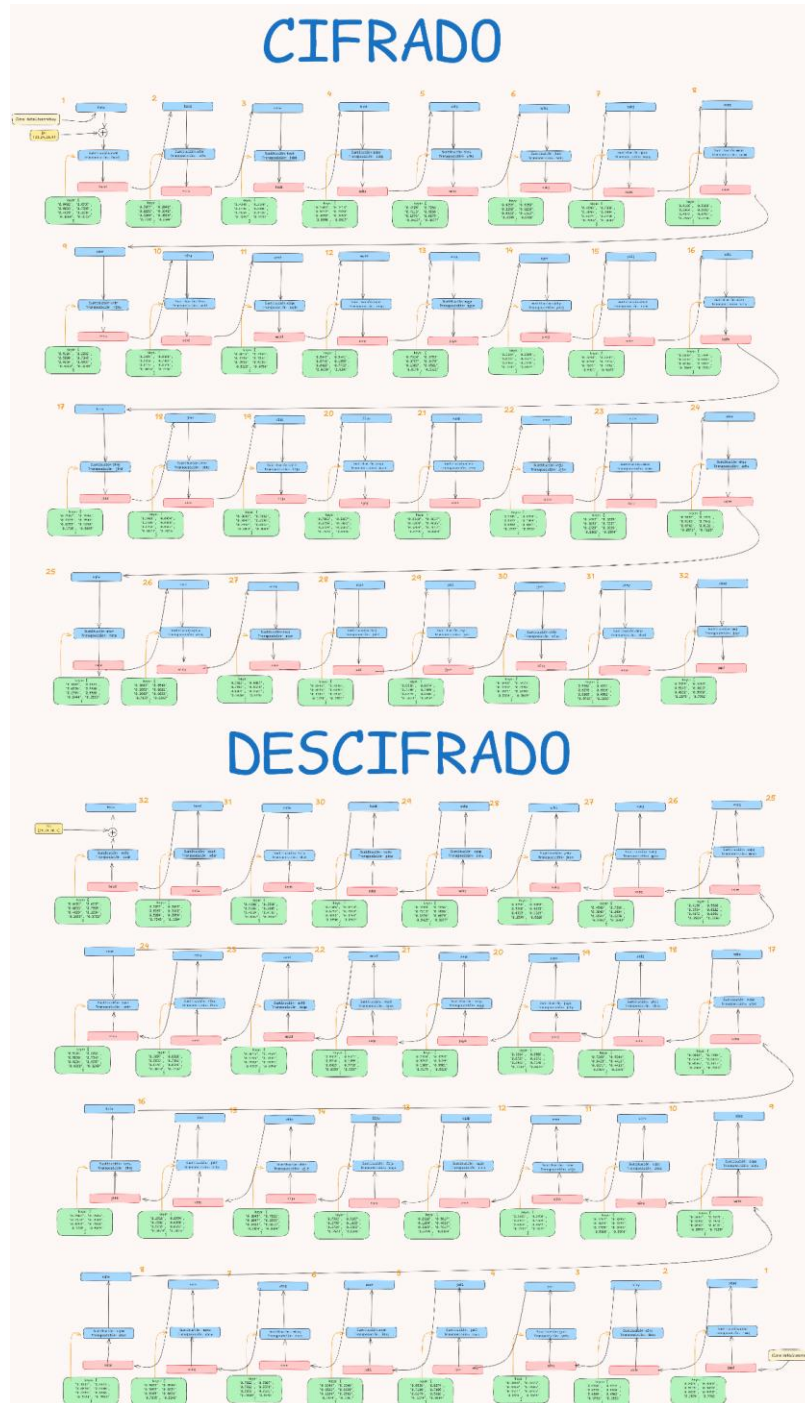
La visualización del descifrado es similar, pero en orden inverso, mostrando cómo se reconstruye el texto original a partir del cifrado.

Diagrama de bloques y repositorio

A continuación, se muestra el algoritmo de forma gráfica para el cifrado de la palabra “hola”, junto con el enlace al repositorio donde se encuentra una imagen en alta calidad y el código fuente.

Figura 31

Diagrama de descifrado y descifrado por lotes



El anterior diagrama es posible encontrarlo con mayor definición en el apéndice F. Mientras que el repositorio al código fuente de este desarrollo se encontrará en el apéndice G

Conclusiones

El desarrollo de este taller permitió verificar la importancia que tienen la seguridad informática y las implicaciones que se deriven de ésta, en diferentes planos, como lo es el técnico, ético y humano. La seguridad informática es fundamental en el marco tecnológico, ya que a medida que las tecnologías evolucionan también lo hacen las amenazas, por lo que es necesario conocer los fundamentos de la protección de la información.

Por otro lado, el control de acceso nos da una idea clara de cómo se puede limitar y en qué niveles de privilegio se puede restringir y/o manejar información. Un ejemplo de esto son los Modelos como el de Bell-LaPadula y Biba, los cuales demuestran cómo garantizar la información tanto confidencial en el modelo de Bell-LaPadula y la integridad en el modelo Biba.

Además, el estudio de las técnicas de cifrado y descifrado de información, como el Método Vigenère y el cifrado por lotes nos demuestra cómo se puede reforzar la seguridad mediante el uso de métodos que transforman los datos.

En conclusión, este taller demuestra que la seguridad informática no solo incluye herramientas tecnológicas, sino que también aborda temas como el diseño de modelos y técnicas para proteger la información, salvaguardando la integridad, confidencialidad y disponibilidad de los datos.

Referencias

- Coronel Suárez, Iván Alberto, & Quirumbay Yagual, Daniel Ivan. (2022). *Seguridad informática, metodologías, estándares y marco de gestión en un enfoque hacia las aplicaciones web*. *Revista Científica y Tecnológica UPSE (RCTU)*, 9(2), 97-109.
<https://doi.org/https://doi.org/10.26423/rctu.v9i2.672>
- Ética, el factor humano más importante en el ámbito de la ciberseguridad. (2016, 20 septiembre). <https://www.welivesecurity.com/la-es/2016/09/20/etica-en-ciberseguridad-factor-humano>
- Gabriela.Bustelo. (2023, 21 septiembre). *Qué son los ciberataques activos y pasivos y cómo evitarlos*. *Red Seguridad*. https://www.redseguridad.com/actualidad/ciberataques-activos-pasivos-que-son-como-evitar_20230921.html#:~:text=El%20ciberataque%20activo%20utiliza%20t%C3%A1cticas,de%20un%20sistema%20o%20red.
- Ibm. (2024, 18 julio). *Seguridad informática*. IBM. <https://www.ibm.com/mx-es/topics/it-security>
- Mks, S. (2023, 20 octubre). *Importancia del Hacking Ético en la Seguridad Informática*. <https://es.linkedin.com/pulse/importancia-del-hacking-%C3%A9tico-en-la-seguridad-inform%C3%A1tica-asgce>
- Novak, J. D., & Cañas, A. J. (2008). *The Theory Underlying Concept Maps and How to Construct and Use Them*. Institute for Human and Machine Cognition (IHMC).
- Silva, Neif, & Espina, Jane. (2006). *Ética Informática en la Sociedad de la Información*. *Revista Venezolana de Gerencia*, 11(36), 559-580. Recuperado en 07 de abril de 2025, de

http://ve.scielo.org/scielo.php?script=sci_arttext&pid=S1315-99842006000400004&lng=es&tlng=es.

Lista de Figuras

Seguridad informática y gestión del riesgo.....	5
Estructura del proyecto de Bell LaPadula.....	5
Clase “acl.py” del proyecto de Bell LaPadula	6
Salida esperada para el caso de: nivel 2 para acceder a un archivo nivel 3.	7
Estructura del proyecto del modelo Biba.....	8
Clase “acl.py” del proyecto de Biba	8
Datos usados en el “Descifrador Afin”	14
Descripción de los colores para los caracteres del texto cifrado y descifrado	14
Algoritmo de cifrado Vigenère base	16
Interfaz del programa del cifrador Vigenère.....	18
Ejemplo del cifrado usando Vigenère.....	19
Función para limpiar los datos.....	20
Función para generar claves.....	20
Función para cifrar por el método Vigenère	21
Función de entrada de los datos por parte del usuario	22
Definición de parámetros usados	23
Función para generar llaves	25
Función para limpiar y dividir el mensaje inicial	26
Operación XOR usando VI o bloque previo.....	27
Función para la sustitución de caracteres.....	28
Función para la transposición del bloque.....	29
Operación para la división del texto cifrado en bloques	29
Operación de transposición inversa	30
Operación de sustitución inversa	31
Operación XOR inversa.....	32
Eliminación de caracteres de relleno	32
Texto original, cifrado y datos usados	33
Resultados de las rondas	34
Diagrama de descifrado y descifrado por lotes.....	35

Lista de Tablas

Matriz de acceso del modelo Bell LaPadula	7
Matriz de acceso para el modelo Biba	9

Apéndices

Apéndice A

Mapa conceptual: Fundamentos y Gestión del Riesgo en Seguridad Computacional

El siguiente mapa conceptual presenta una síntesis visual de los principales conceptos relacionados con la seguridad de la información. Puede consultarse en el siguiente enlace:

<https://drive.google.com/file/d/1nhZDLtQCDHEaK-lvHRHbQimmg4FL8KuT/view>

Apéndice B

Repositorio: Modelos de control de acceso Bell LaPadula y Biba

El siguiente repositorio presenta los modelos de Bell LaPadula y Biba. Puede consultarse en el siguiente enlace: <https://github.com/Edwin1016-rgb/modelosDeControlDeAcceso>

Apéndice C

Documento: procedimiento para el descifrador afín.

En la siguiente plantilla de Excel se presenta lo necesario para lograr el descifrador afín. Puede consultarse en el siguiente enlace:

<https://docs.google.com/spreadsheets/u/0/d/1rNkSKcjaWdgtC9KN-EesHKOypY970-OB/edit>

Apéndice D

Repositorio: algoritmo para el descifrador afín.

En el siguiente repositorio se presenta el código de apoyo referente al descifrador Afín.

Puede consultarse en el siguiente enlace: https://github.com/Cr1ss4nB/Afin_Decipher

Apéndice E

Repositorio: codificador de Vigenere

En el siguiente repositorio se presenta el código referente al codificador de Vigenere.

Puede consultarse en el siguiente enlace: https://github.com/Cr1ss4nB/Vigen-re_Encoder

Apéndice F

Diagrama: diagrama del cifrador y descifrador de bloques

En el siguiente repositorio se encuentra la imagen correspondiente al diagrama de cifrado y descifrado de bloques. Puede consultarse en el siguiente enlace:

https://github.com/gabo8191/cipher_algorithm/blob/main/diagrams/cipher_decipher.png

Apéndice G

Repositorio: código de ciframiento-desciframiento por bloques.

En el siguiente repositorio se encuentra el código correspondiente al cifrador-descifrador de bloques. Puede consultarse en el siguiente enlace:

https://github.com/gabo8191/cipher_algorithm