Laboratorio 2: Tel252

Javiera Fuentes, Nicolás Vergara, Pablo Gajardo, Camilo Muñoz y Vicente Moya

August 26, 2024

1 Introducción

1.1 Contexto

En el estudio de la criptografía es fundamental comprender el concepto de algoritmos clásicos, ya que esto permitira analizar y romper sistemas criptograficos simples. Debido a lo anterior el laboratorio se enfoca en utilizar SageMath para la implementación de algoritmos clásicos, tales como el cifrado César, el cifrado Afín y el cifrado Hill.

1.2 Objetivos

- **Usar SageMath para desarrollar algoritmos de cifrado y descifrado:** Los estudiantes desarrollarán habilidades en el uso de esta plataforma para resolver problemas criptográficos.
- **Comprender los métodos de cifrado clásicos y descifrado:** Los estudiantes comprenden el funcionamiento interno de estos métodos de cifrado. Esto implica entender cómo se manipulan los datos y los principios matemáticos subyacentes.
- **Implementar métodos de cifrado clásicos y descifrado:** Los estudiantes aprenderan a implementar varios métodos de cifrado clásicos a partir del entendimiento de la teoría. Esto les permitirá familiarizarse con los fundamentos de la criptografía y entender cómo se utilizaban para proteger información.

El código generado para esta experiencia puede ser encontrado en el siguiente repositorio de github.

2 Metodología

2.1 Flujo del laboratorio

- En primer lugar, se muestran los ejercicios con una breve explicacion por parte del profesor y del ayudante. Además, se menciona el formato de entrega de este laboratorio, siendo la primera parte un entregable en la misma clase y la segunda parte para el siguiente domingo.
- Comenzado el trabajo en grupos, se trabaja en la parte a) de la tercera pregunta. En esta parte se nos pide una funcion de ataques de frecuencia que entregue una lista de pares ordenada con el porcentaje de aparicion de cada caracter en el texto cifrado. Se da por finalizada en clase esta parte de la problematica.
- En la parte b) se solicita una funcion capaz de traducir por partes un texto cifrado. Para esto, se utiliza un llamado a primera funcion y se reemplazan los guiones bajos de un subtexto con cada letra que haya sido descifrada por el equipo. Esta parte del laboratorio no pudo ser completada durante la clase.
- Para finalziar la experiencia, se enviaron los archivos desarrollados durante el desarrollo del laboratorio antes del final del bloque.

2.2 Ejercicios desarrollados durante la clase

Aqui se describen algunas funciones realizadas en clases. notar que en la sesscion 3 se re hicieron por completo, optimizando lo hecho en el laboratorio.

- Implementa una funcion de Sage que realice ataques de frecuencia en cifrados de sustitucion mono- alfabetica

```
    """
    Funcion que crea un diccionario inicializando la cantidad de letras del alfabeto
    ingles en cero, retorna una lista de tuplas ordendas de mayor a menor, donde se
    tiene el promedio de cada letra.
    """
def frecuencias(palabra):
    veces = {}
    for _ in range(26):
        if(not chr(97+_) in veces ):
            veces[chr(97 + _)] = 0

    cantidad = 0
    for letra in palabra:
        if(ord(letra.lower())>=97 and ord(letra.lower())<=122):
            veces[letra.lower()]+=1
            cantidad+=1

    histograma = []
    for letra,vez in veces.items():
        numero = int(vez)/int(cantidad)
        veces[letra] = numero
        histograma.append((letra,numero))
    return sorted(histograma, key=lambda x: x[1],reverse=True)

```

- Implementa una funcion de Sage que tome una sustituci´on mono-alfab´etica parcial y un texto cifrado, y devuelva un posible texto plano

```
    """
    Funcion 'cambio' toma las frecuencias de cada letra del texto cifrado y las cambia
    por las dos letrasmas utilizadas del alfabeto ingles, mientras que el resto
    lo rellena con '_'.
    """
def cambio(palabra):

    cambio = []
    frecuencia = frecuencias(palabra)

    for letra in palabra:
        if(letra == frecuencia[0][0]):
            cambio.append('e')
        if(letra == frecuencia[1][0]):
            cambio.append('t')
        else:
            cambio.append('_')
    return ''.join(cambio)

```

- Usa tus funciones de (a) y (b) para descifrar.
  Aqui se uso para testear una palabra aleatoria (vista en el laboratorio 1), la cual es gryy gurz gb tb gb nzoebfr puncry. Se obtuvo lo siguiente :

```

palabra = 'gryy gurz gb tb gb nzoebfr puncry'

#===Frecuencias
print(frecuencias(palabra))

#===Texto unido
print(cambio('gryy gurz gb tb gb nzoebfr puncry'))

#tell them to go to ambrose chapel. ===> texto original descifrado.


    # Resultado en el terminal:
[('b', 0.14814814814814814), ('g', 0.14814814814814814), ('r', 0.14814814814814814),
('y', 0.1111111111111111), ('n', 0.07407407407407407), ('u', 0.07407407407407407),
('z', 0.07407407407407407), ('c', 0.03703703703703035), ('e', 0.03703703703703035),
('f', 0.03703703703703035), ('o', 0.03703703703703035), ('p', 0.03703703703703035),
('t', 0.03703703703703035), ('a', 0.0), ('d', 0.0), ('h', 0.0), ('i', 0.0), ('j', 0.0),
('k', 0.0), ('l', 0.0), ('m', 0.0), ('q', 0.0), ('s', 0.0), ('v', 0.0), ('w', 0.0),
('x', 0.0)]

t____t____te___e__te______e__________
```

  Notar que el texto devuelto por la funciones es `t____t____te___e__te______e__________`, parecido al original que es `tell them to go to ambrose chapel`.

3 Ejercicios Prácticos

3.1 Pregunta 2

Implementa funciones de Sage que realicen el cifrado/descifrado afín, dado una clave que consiste en un par de enteros (a, b), ambos en 1, 2, ..., 25 con a no divisible por 2 o 13. Las funciones deben trabajar con cadenas de texto y dejar cualquier carácter no alfabético sin cambios. Muestra la operación de tus funciones en un ejemplo.

- $k = (17, 8)$, texto plano = "john smith is the culprit!"

1. Función de cifrado

```
    """
    funcion de cifrado afin, recibe un texto plano y una llave la cual es una tupla con
    dos terminos (a,b)
    """
    def cifrado_afin(k, texto):
        a , b = k
        texto_cifrado = []

        for letra in texto:
            if(ingles_alfabeto(letra)):
                if(verificar_a(a)):
```

```
                    c = mod((a*convertir_numero(letra))+b)
                    texto_cifrado.append(chr(c+97))
                else:
                    print("El valor de 'a' es invalido")
            else:
                texto_cifrado.append(letra)

        return ''.join(texto_cifrado)

        """Prueba cifrado afin"""

    print(cifrado_afin((17,8),"john smith is the culprit!"))

    # Resultado en el terminal:

    fmxv ceotx oc txy qkndlot!
```

2. Función de descifrado

```
    """
    funcion de descifrado afin, recibe un texto cifrado y una llave la cual es una tupla
    con dos terminos (a,b)
    """
    def descifrado_afin(k, texto_cif):
    a , b = k
    texto_original = []

    for letra in texto_cif:
        if(ingles_alfabeto(letra)):
            if(verificar_a(a)):
                p = mod((convertir_numero(letra)-b)/a)
                texto_original.append(chr(p+97))
            else:
                print("El valor de 'a' es invalido")
        else:
            texto_original.append(letra)
    return ''.join(texto_original)

    """Prueba descifrado afin"""

    print(descifrado_afin((17,8),"fmxv ceotx oc txy qkndlot!"))

    # Resultado en el terminal:

    john smith is the culprit!
```

3.2 Pregunta 3

1. Implementa una funcion de Sage que realice ataques de frecuencia en cifrados de sustitucion mono- alfabetica.

   Primero creamos una funcion que maneje las frecuencias de los caracter de un texto cifrado.

```
    """
    Funcion frecuncias recibe un texto cifrado, tal que inicializa un diccionario con
    todas las letras del alfabeto ingles, luego va por el texto y suma caracter por
    caracter, cada vez que se repite. Luego crea una lista de tuplas que se
    ordena de mayor a menor. Finalmente retorna la lista ordenada.
    """

def frecuencias(palabra):
    veces = {
        chr(97 + _):0 for _ in range(26)
    }
    cantidad = 0

    for letra in palabra:
        if (ord(letra.lower()) >= 97 and ord(letra.lower()) <= 122):
            veces[letra.lower()] += 1
            cantidad += 1

    histograma = [ (letra, int(vez) / int(cantidad)) for letra, vez in veces.items()]

    return sorted(histograma, key=lambda x: x[1],reverse=True)
    # Resultado en el terminal:


```

2. Implementa una funcion de Sage que tome una sustitucion mono-alfabetica parcial y un texto cifrado, y devuelva un posible texto plano.

   Para esto se implemento la siguiente funcion sustitucion.

```
    """
    Funcion sustitucion obtiene las frecuencias de la funcion del item anterior, y dada
    la tabla de letras con mayor frecuencias del alfabeto ingles, reemplaza las que mas
    se repiten del texto cifrado, mientras que las demas se cambia por '_'.
    """

def sustitucion(palabra, frecuencia):

    #frecuencia = frecuencias(palabra)
    cambio = [
        'e' if letra == frecuencia[0][0]
        else 't' if letra == frecuencia[1][0]
        else 'a' if letra == frecuencia[2][0]
        else '_'
        for letra in palabra
    ]
    return ''.join(cambio)
```

Luego se implementa una segunda funcion que toma lo arrojado por la anterior y en los espacios con guiones, coloca la letra en mayuscula del mensaje cifrado, mientras que las letras mas frecuentes las mantiene.

```
def union_final(palabra,descifrado):

    union = [
            palabra[_].upper()
            if descifrado[_] == '_' else descifrado[_]
            for _ in range(len(palabra))
            ]
    return ''.join(union)
```

3. Usa tus funciones de (a) y (b) para descifrar el siguiente texto cifrado:

   “ztmn pxtne cfa peqef kecnp cjt tmn zcwsenp ontmjsw ztnws tf wsvp xtfwvfefw, c feb fcwvtf, xtfxevqea vfgvoenwk, cfa aeavxcwea wt wse rntrtpvwvtf wscw cgg lef cne xnecwea eymcg.”

   Se implemento la el siguiente es

```

#====Frecuencias uso de funcion frecuencias, item 1
print('Frecuencias. \n')
print(frecuencias(palabra))

#====Sustitucion uso de funcion sustitucion, item 2

print('\nSustitucion. \n')
print(sustitucion(palabra,frecuencias(palabra)))

#====Union final uso de funcion union final, item 2

print('\nUnion final. \n')
print(union_final(palabra,sustitucion(palabra,frecuencias(palabra))))

    # Resultado en el terminal:

Frecuencias.

[('e', 0.1258741258741259), ('w', 0.1048951048951049), ('f', 0.0979020979020979),
('t', 0.0979020979020979), ('c', 0.09090909090909091), ('n', 0.07692307692307693),
('v', 0.06293706293706294), ('a', 0.04895104895104895), ('p', 0.04195804195804196),
('s', 0.04195804195804196), ('x', 0.04195804195804196), ('g', 0.027972027972027972),
('m', 0.027972027972027972), ('z', 0.02097902097902098), ('j', 0.013986013986013986),
('k', 0.013986013986013986), ('o', 0.013986013986013986), ('q', 0.013986013986013986),
('r', 0.013986013986013986), ('b', 0.006993006993006993), ('l', 0.006993006993006993),
('y', 0.006993006993006993), ('d', 0.0), ('h', 0.0), ('i', 0.0), ('u', 0.0)]

Sustitucion.

_________e__a___e_ea__e_______________t_e_________t____t___a_t______at_aeat____ae__
a_t__a____a_e__e___a___e_t____a___e____te__t__t_e________t__a_t__t______ea___e___
e_te__e_____

```

```
Union final.

ZTMN PXTNe CaA PeQea KeCNP CJT TMN ZCtSeNP ONTMJSt ZTNtS Ta tSVP XTatVaeat, C aeB
↪ aCtVTa,
XTaXeVQeA VaGVOeNtK, CaA AeAVXCteA tT tSe RNTRTPVtVTa tSCt CGG Lea CNe XNeCteA eYMCG.
```

3.3 Pregunta 4

Implementa funciones de Sage para realizar cifrado/descifrado con el Cifrado de Hill 2x2. La clave debe ser una matriz invertible de Sage sobre los enteros módulo 26. No solo llames a la funcionalidad incorporada de Sage para el cifrado de Hill. Muestra la operación de tus funciones en un texto plano de tu elección.

1. Función de cifrado

```
    """
    funcion de cifrado hill, recibe un texto plano y una llave la cual es una matriz de
    2x2 invertible de Sage sobre los enteros modulo 26.
    """
    def cifrado_hill_2x2(texto, K):
        key = Matrix(ZZ,K)
        pares = []

        for i in range(0, len(texto), 2):
            pares.append(texto[i:i+2])

        texto_cifrado = []

        if(key.is_invertible()):
            for par in pares:
                vector = matrix([[convertir_numero(par[0])],
                ↪ [convertir_numero(par[1])]])
                c = key * vector
                for i in range (0,2):
                    coef = mod(int(c[i,0]))
                    texto_cifrado.append(chr(coef+97))
        else:
            print("La matriz clave no es invertible, ingrese otra")
        return ''.join(texto_cifrado)

    """Prueba cifrado hill 2x2"""

    print(cifrado_hill_2x2("ilikepurpledinosaurs", [[1,2], [3,5]]))

    # Resultado en el terminal:

    ebcwijcplwkbilycowbl
```

2. Función de descifrado

```
    """
```

```
    funcion de descifrado hill, recibe un texto cifrado y una llave la cual es una
    ↪ matriz
    de 2x2 invertible de Sage sobre los enteros modulo 26.
    """
    def descifrado_hill_2x2(texto_cif, K):
        key = Matrix(ZZ,K)

        pares = []

        for i in range(0, len(texto_cif), 2):
            pares.append(texto_cif[i:i+2])

        texto_original = []

        if(key.is_invertible()):
            inversa = key.inverse()
            for par in pares:
                vector = matrix([[convertir_numero(par[0])],
                ↪ [convertir_numero(par[1])]])
                c = inversa * vector
                for i in range (0,2):
                    coef = mod(int(c[i,0]))
                    texto_original.append(chr(coef+97))
        else:
            print("La matriz clave no es invertible, ingrese otra")
        return ''.join(texto_original)

    """Prueba descifrado hill 2x2"""

    print(descifrado_hill_2x2("ebcwijcplwkbilycowbl", [[1,2], [3,5]]))

    # Resultado en el terminal:

    ilikepurpledinosaurs

```

3.4 Pregunta 5

Implementa una función de Sage para realizar cifrado/descifrado con el Cifrado de Hill mxm. La clave debe ser una matriz invertible de Sage sobre los enteros módulo 26. No solo llames a la funcionalidad incorporada de Sage para el cifrado de Hill. Muestra la operación de tu función en las funciones que escribas sobre un texto plano de tu elección. Puedes usar cualquier función que hayas escrito para la pregunta anterior para responder esta pregunta.

1. Función de cifrado

```
    '''
    funcion de cifrado de hill, toma una matriz m*m como argumeto (en lugar de una de
    ↪ 2x2) en conjunto con el texto a cifrar
    '''
    def cifrado_hill_mxm(texto_cif, matriz):

        K = Matrix(ZZ, matriz)
```

```
        palabras_por_particion = K.nrows()
        cant_particiones = math.ceil(len(texto_cif) / palabras_por_particion)

        nletra = 0
        particiones = []
        vectores = []
        vector = []
        vector_cifrado = []
        resultado = 0
        resultado_mod = 0

        char1 = ""
        mensaje_cif = ""

        if(K.determinant() == 0 or (K.determinant()%2 == 0 or K.determinant()%13 == 0 or
        ↪  K.determinant()%26 == 0)):
            print("La Matriz llave no es invertible, o no es invertible en el modulo
            ↪  26")
            return 0

        for i in range(cant_particiones):
            particiones.append(texto_cif[i*palabras_por_particion:
            ↪  (i+1)*palabras_por_particion])

        if(len(particiones[cant_particiones - 1]) != palabras_por_particion):

            particiones[cant_particiones - 1] += "x"*(palabras_por_particion -
            ↪  len(particiones[cant_particiones - 1]))

        for i in range(cant_particiones):

            for j in range(len(particiones[i])):

                nletra = convertir_numero(particiones[i][j])
                vector.append(nletra)

            vectores.append(vector)
            vector = []

        vectores = Matrix(ZZ, vectores)

        for i in range(vectores.nrows()):

            resultado = (vectores[i]*K)
            resultado_mod = mod(resultado)
            vector_cifrado.append(resultado_mod)

        for i in range(cant_particiones):

            for j in range(palabras_por_particion):

```

```
                char1 = chr(vector_cifrado[i][j] + 97)
                mensaje_cif = mensaje_cif + char1

        return mensaje_cif

    '''mostrar texto cifrado '''
    mensaje_cifrado = cifrado_hill_mxm("lastresveintedelamanana", [[6, 24, 1],[13, 16,
    ↪  4],[5, 9, 17]])
    print(mesaje_cifrado)

    # Resultado en el terminal:
    akfrkzlyoaztfygwihnaalzo
```

2. Función de descifrado

```
    '''
    funcion de descifrado de hill. toma como argumentos el string de texto crifrado y la
    ↪  matriz que se usó en la matriz de cifrado anteriormente
    '''
    def descifrado_hill_mxm(texto_cif, matriz):

        K = Matrix(ZZ, matriz)
        palabras_por_particion = K.nrows()
        cant_particiones = math.ceil(len(texto_cif) / palabras_por_particion)

        nletra = 0
        particiones = []
        vectores = []
        vector = []
        vector_descifrado = []
        resultado = 0
        resultado_mod = 0

        char1 = ""
        mensaje_descif = ""

        if(K.determinant() == 0 or (K.determinant()%2 == 0 or K.determinant()%13 == 0 or
        ↪  K.determinant()%26 == 0)):
            print("La Matriz llave no es invertible, o no es invertible en el modulo
            ↪  26")
            return 0

        for i in range(cant_particiones):
            particiones.append(texto_cif[i*palabras_por_particion:
            ↪  (i+1)*palabras_por_particion])

        for i in range(cant_particiones):

            for j in range(len(particiones[i])):

                nletra = convertir_numero(particiones[i][j])
                vector.append(nletra)
```

```
34
35              vectores.append(vector)
36              vector = []
37
38          vectores = Matrix(ZZ, vectores)
39
40          for i in range(vectores.nrows()):
41
42              resultado = (vectores[i]*K.inverse())
43              resultado_mod = mod(resultado)
44              vector_descifrado.append(resultado_mod)
45
46          for i in range(cant_particiones):
47
48              for j in range(palabras_por_particion):
49
50                  char1 = chr(vector_descifrado[i][j] + 97)
51                  mensaje_descif = mensaje_descif + char1
52
53          return mensaje_descif
54
55      '''prueba descifrado hill m*m'''
56      print(descifrado_hill_mxm(mensaje_cifrado, [[6, 24, 1],[13, 16, 4],[5, 9, 17]]))
57
58      # Resultado en terminal:
59      lastresveintedelamananax
60
```

3.5 Pregunta 6

Esta pregunta es para implementar y usar un ataque de texto plano conocido en el cifrado de Hill. Puedes usar funciones de ejemplos o preguntas anteriores, pero no uses las funciones incorporadas de Sage para el cifrado de Hill.

```
1   def ataque_cifrado_hill(texto_plano, texto_cif):
2
3       palabras_por_particion = 2
4       cant_particiones = math.ceil(len(texto_cif) / palabras_por_particion)
5       particiones_cif = []
6       particiones_pl = []
7
8       nletracif = 0
9       nletrapl = 0
10      a = 0
11      fila_cif = []
12      fila_pl = []
13      matriz_cif = []
14      matriz_pl = []
15
16      mcif_inv = []
17      mpl_inv = []
18      Key = []
19      posibles_k = []
```

```
20
21          for i in range(cant_particiones):
22              particiones_cif.append(texto_cif[i*palabras_por_particion:
                ↪  (i+1)*palabras_por_particion])
23              particiones_pl.append(texto_plano[i*palabras_por_particion:
                ↪  (i+1)*palabras_por_particion])
24
25          for i in range(cant_particiones):
26
27
28              for j in range(len(particiones_cif[i])):
29
30
31                  if(a >= len(texto_plano)):
32
33                      if(a%2 != 0):
34                          fila_pl.append(23)
35
36                      else:
37                          fila_pl.append(23)
38
39                  else:
40
41                      nletrapl = convertir_numero(particiones_pl[i][j])
42                      fila_pl.append(nletrapl)
43
44                  nletracif = convertir_numero(particiones_cif[i][j])
45                  fila_cif.append(nletracif)
46
47                  a += 1
48
49              matriz_cif.append(fila_cif)
50              fila_cif = []
51
52              matriz_pl.append(fila_pl)
53              fila_pl = []
54
55
56          matriz_pl = Matrix(ZZ, matriz_pl)
57          matriz_cif = Matrix(ZZ, matriz_cif)
58
59          for i in range(cant_particiones):
60
61              mcif_inv = matriz_cif[i:i+2]
62              mpl_inv = matriz_pl[i:i+2]
63
64              if (mcif_inv.nrows() == 2):
65
66                  determinante_pl = mpl_inv.det()
67                  determinante_cif = mcif_inv.det()
68
```

```
69
70                  if((determinante_cif != 0 and determinante_pl != 0) and
                    ↪  (mpl_inv.determinant()%2 != 0 and mpl_inv.determinant()%13 != 0) and
                    ↪  (mcif_inv.determinant()%2 != 0 and mcif_inv.determinant()%13 != 0)):
71
72                      det_inverso = (1/mpl_inv.determinant())%26
73
74                      mpl_inv = (det_inverso * mpl_inv.adjugate()%26)%26
75
76                      Key = mpl_inv*mcif_inv
77                      posibles_k.append(Matrix(ZZ,(Key%26)))
78
79          return posibles_k
80
81      '''Pruebas de ataques'''
82
83      #Resultado en el terminal
84      print(ataque_cifrado_hill("friday", "izrvey"))
85
86      [[11 12]
87      [11  1]]
88
89      print("")
90
91      print(ataque_cifrado_hill("diamondisinstatue", "zisxlhdiwdingthyqq"))
92
93      [[11 20]
94      [19 13]]
95
96      print("")
97
98      print(ataque_cifrado_hill("hesecretdietistofuhotdogs", "qbayzelwilksscipqpsvkafvssyy"))
99
100     [[14 19]
101     [25  4]]
102
103     print("")
```

4 Conclusiones

La implementación de algoritmos de cifrado y descifrado en SageMath ha proporcionado una visión clara sobre técnicas criptográficas clásicas. En particular, el cifrado afín y el cifrado de Hill han sido abordados con éxito, demostrando cómo se pueden aplicar estas técnicas para transformar y recuperar información.

Para el cifrado afín, se desarrollaron funciones que permiten cifrar y descifrar textos utilizando una clave de dos enteros. La clave debe cumplir con la condición de que uno de los parámetros no sea divisible por 2 o 13 para asegurar la validez del cifrado. Las pruebas realizadas confirmaron que el cifrado y descifrado funcionan correctamente, transformando y recuperando el texto según lo esperado.

En el caso del cifrado de Hill, se implementaron versiones de 2x2 y mxm. La función para el cifrado Hill 2x2 utiliza una matriz 2x2 para cifrar pares de caracteres, mientras que la versión mxm maneja

matrices de mayor tamaño. Se comprobó que las funciones operan correctamente siempre que la matriz clave sea invertible en el módulo correspondiente, y las pruebas confirmaron que tanto el cifrado como el descifrado son efectivos.

4.1 Análisis general

El desarrollo del laboratorio de cifradores clásicos permitió una comprensión profunda de los fundamentos criptográficos y su implementación práctica mediante el uso de SageMath. A lo largo de las actividades, se exploraron diversas técnicas de cifrado y descifrado, como los cifrados de César, Afín y Hill. Cada uno de estos métodos tiene aplicaciones históricas y pedagógicas que refuerzan la comprensión de los conceptos criptográficos básicos.

Durante la implementación de los algoritmos, se evidenció la importancia de comprender no solo los procedimientos matemáticos detrás de cada cifrado, sino también las implicaciones de seguridad que conllevan. Por ejemplo, la facilidad con la que un cifrado de César puede ser vulnerado mediante un ataque de fuerza bruta demostró la necesidad de utilizar métodos más sofisticados en aplicaciones reales.

El ejercicio de descifrado utilizando ataques de frecuencia y ataques de texto plano conocido no solo reafirmó la vulnerabilidad de los cifrados clásicos, sino que también subrayó la evolución hacia métodos más seguros y complejos en la criptografía moderna. Este laboratorio, por tanto, sirvió como una base sólida para comprender la transición desde los cifrados clásicos a técnicas criptográficas más avanzadas.

4.2 Reflexiones

El laboratorio no solo cumplió con los objetivos planteados, sino que también brindó una valiosa experiencia en la implementación práctica de conceptos teóricos. La oportunidad de experimentar con distintos cifradores permitió visualizar de manera clara las fortalezas y debilidades de cada método.

Una reflexión clave es la relevancia continua de los conceptos básicos de la criptografía, incluso en un contexto moderno. Aunque los cifradores clásicos ya no se utilizan en la protección de información sensible, su estudio es crucial para comprender la evolución y los principios fundamentales de la criptografía moderna.

Asimismo, la experiencia de realizar ataques criptográficos, como el ataque de frecuencia y el ataque de texto plano conocido, resalta la importancia de desarrollar y emplear algoritmos criptográficos robustos que puedan resistir estos tipos de análisis. Este entendimiento es esencial para cualquier profesional que aspire a trabajar en el campo de la seguridad informática.

En conclusión, el laboratorio proporcionó no solo conocimientos técnicos, sino también una apreciación crítica de la criptografía como disciplina. Las actividades realizadas subrayan la importancia de una sólida comprensión teórica acompañada de habilidades prácticas para enfrentar los desafíos contemporáneos en seguridad de la información.

References

[1] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 7th ed. London, UK: Pearson, 2017.