Universidad Nacional Autónoma de México
Facultad de Ciencias

Facultad de CIencias

Criptografía y Seguridad

Semestre: 2025-1

Equipo: Pingüicoders

Práctica 4:
Cifrados Clásicos

Arrieta Mancera Luis Sebastián - 318174116
Cruz Cruz Alan Josue - 319327133
García Ponce José Camilo - 319210536
Matute Cantón Sara Lorena - 319331622

Introducción:

Desde los inicios ha sido naturaleza humana querer comunicar de un modo seguro información la cual debe mantenerse secreta, esto ha culminado en la creación de diversos cifrados. En esta práctica solo se van a ver 3 cifrados clásicos para entender su comportamiento y generar un programa capaz de descifrar y cifrar la información. Aunque en la actualidad estos 3 no se usen, conocer y entender cómo funcionan sirve como una introducción al mundo de la criptografía y como estos se apoyan de modelos matemáticos para su funcionamiento, al igual que nos proporcionan una herramienta para enfrentarse a cifrados más complejos.

Desarrollo:

Lo primero que decidimos hacer para resolver el problema de la práctica fue hacer un método para resolver dos congruencias. Para esto usamos las siguientes metodos:

+ Método para obtener el Maximo Comun Divisor

```python
def mcd(a,b):
    """
    Regresa el maximo comun divisor de dos numeros.
    """
    while b != 0:
        a = a % b
        # Swap
        aux = a
        a = b
        b = aux
    return a
```

+ Método para reducir un número x a módulo n

```python
def reduce(x,n):
    """
    Regresa un numero reducido a modulo n
    """
    negativo = x < 0
    numero_reducido = abs(x) % n
    return n - numero_reducido if negativo else numero_reducido
```

+ Método para obtener el inverso de un número x en módulo n

```python
def inverso(x,n):
    """
    Regresa el inverso multiplicativo de un numero (si lo hay).
    """
    # Un numero tiene inverso multiplicativo si x,n son primos relativos (coprimos)
    if mcd(x,n) == 1:
        a, s, t = euclides_extendido(x,n)
        return reduce(s,n)
    else:
        print("No existe el inverso multiplicativo de", x, "mod", n, " nos dio ", mcd(x,n))
        raise ValueError(f"El inverso multiplicativo de {x} mod {n} no existe")
```

+ Método del Algoritmo de Euclides extendido

```python
def euclides_extendido(a, b):
    """
    Regresa el maximo comun divisor de 'a' y 'b' junto con los coeficientes de la combinación lineal.
    """
    # Combinacion lineal
    # r = sa + tb
    s_0 = 1     # Coeficiente s
    s_1 = 0     # Carga con el divisor
    t_0 = 0     # Coeficiente t
    t_1 = 1     # Carga con el dividendo
    while b != 0:
        # Algoritmo de la division
        # a = bq + r
        q = int(a / b)  # Cociente
        r = a % b       # Residuo
        # Swap
        a = b
        b = r
        # Actualizamos los coeficientes s y t (Swap)
        s_0, s_1 = s_1, s_0 - q * s_1
        t_0, t_1 = t_1, t_0 - q * t_1
    # Imprimimos la combinacion lineal: r = as + bt
    # print(f"{q} = {x}({s_0}) + {y}({t_0})")
    # print(a,b,s_0,s_1,t_0,t_1)
    return a, s_0, t_0
```

Y con esos métodos pudimos crear nuestro método para resolver congruencias (con los pasos vistos en el laboratorio)

```python
# Metodo para resolver dos congruencias de la forma ax + y ≡ b (mod m) y cx + y ≡ d (mod m), obteniendo los valores de x y y
def resolver_congruencias(a, b, c, d, m):
    # Primero resolver para x
    # Restar la segunda congruencia de la primera para eliminar y
    # (ax + y - (cx + y)) ≡ (b - d) (mod m) => (a - c)x ≡ (b - d) (mod m)
    a_menos_c = a - c
    b_menos_d = b - d
    # Encontrar x en (a - c)x ≡ (b - d) (mod m)
    # Encontrar el inverso modular de (a - c) mod m
    inv = inverso(a_menos_c % m, m)
    # Resolver para x
    x = (inv * (b_menos_d % m)) % m
    # Resolver para y
    # Sustituir x en ax + y ≡ b (mod m) y encontrar y
    y = (b - a * x) % m
    return x, y
```

Después investigamos (en internet y viendo archivos propios) los bytes mágicos para los archivos png, pdf, mp4 y mp3, y los pusimos en forma decimal

```python
# Bytes Magicos
bpng = [137, 80, 78, 71, 13, 10, 26, 10]
bpdf = [37, 80, 68, 70]
bmp4 = [0, 0, 0, 32, 102, 116, 121, 112]
bmp3 = [73, 68, 51]
bytes_magicos = [bpng, bpdf, bmp4, bmp3]
```

Luego creamos un método para que dado un archivo lo intente descifrar (ver si es un png, pdf, mp4 o mp3) y guarde el archivo descifrado, pero para esto usamos algunos métodos extras, los cuales son:

+ Método para obtener los 8 primeros bytes de un archivo (en forma decimal)

```python
# Metodo para abrir un archivo y obtener los primeros 8 bytes en decimal
def obtener_primeros_8_bytes(archivo):
    with open(archivo, "rb") as f:
        primeros_bytes = f.read(8)
        primeros_numeros = [int(byte) for byte in primeros_bytes]
    return primeros_numeros
```

Método para aplicar una función a todos los bytes de un archivo y guardar el resultado en otro archivo

```python
# Metodo para aplicar una funcion a cada byte de un archivo (en forma de decimal) y guardar el resultado en un nuevo archivo (en forma de bytes)
def aplicar_funcion_a_bytes(archivo, funcion, archivo_salida):
    with open(archivo, "rb") as f:
        bytes = f.read()
        numeros = [int(byte) for byte in bytes]
    resultado = [funcion(byte) % 256 for byte in numeros]
    with open(archivo_salida, "wb") as f:
        f.write(bytearray(resultado))
```

Error por si no se puede descifrar

```python
# Error para devolver cuando no se pueda descifrar un archivo
class ErrorDescifrado(Exception):
    def __init__(self, mensaje):
        super().__init__(mensaje)
```

Método que usando los primeros bytes de un archivo busca una función para que aplicada a los primeros bytes nos de los bytes mágicos de un tipo de archivo. Notemos una cosa interesante aquí, para los bytes mágicos de un mp4 ignoramos los 4 primeros bytes, esto lo pusimos así debido a que un inicio esto no lo hacíamos y no logramos descifrar el mp4, entonces estuvimos revisando varios mp4 que teníamos y notamos que siempre empezaban con 3 ceros y un número que a veces cambiada, por lo tanto decidimos ignorar los 4 primeros bytes y logramos que ya funcionara para mp4s

```python
# Metodo para dado los primeros 8 bytes (en forma decimal) de un archivo, intentar descifrarlo (sabiendo que es un archivo PNG, PDF, MP4 o MP3)
# Devolver una funcion que lo descifre y que tipo de archivo es
def obtener_funcion_descifrado(primeros_bytes):
    for byte_magico in bytes_magicos:
        try:
            print(primeros_bytes)
            print(byte_magico)
            print("---")
            # Caso especial, si son los bytes magicos de un archivo mp4 tomar los bytes 5, 6 y 7
            if byte_magico == bmp4:
                a, b = resolver_congruencias(primeros_bytes[6], byte_magico[6], primeros_bytes[5], byte_magico[5], 256)
                if (a * primeros_bytes[4] + b) % 256 == byte_magico[4]:
                    return lambda z: (z * a + b) % 256, bytes_magicos.index(byte_magico)
            else:
                x, y = resolver_congruencias(primeros_bytes[0], byte_magico[0], primeros_bytes[1], byte_magico[1], 256)
                if (x * primeros_bytes[2] + y) % 256 == byte_magico[2]:
                    return lambda z: (z * x + y) % 256, bytes_magicos.index(byte_magico)
        except:
            pass
    return ErrorDescifrado("No se pudo descifrar el archivo")
```

Y con esos métodos pudimos generar el método que recibe un archivo y lo intenta descifrar y guardar ya bonito

```python
# Metodo para intentar descifrar un archivo y guardar el resultado en un nuevo archivo
def magia(archivo):
    try:
        primeros_bytes = obtener_primeros_8_bytes(archivo)
        funcion_descifrado, tipo_archivo = obtener_funcion_descifrado(primeros_bytes)
        salida = archivo[:-4]
        if tipo_archivo == 0:
            salida += ".png"
        elif tipo_archivo == 1:
            salida += ".pdf"
        elif tipo_archivo == 2:
            salida += ".mp4"
        elif tipo_archivo == 3:
            salida += ".mp3"
        aplicar_funcion_a_bytes(archivo, funcion_descifrado, salida)
    except ErrorDescifrado as e:
        print(e)
```

De esta forma ejecutamos magia("archivos/file2.lol"), magia("archivos/file3.lol") y magia("archivos/file4.lol"), generando un archivo pdf del file2, un archivo mp3 del file3 y un mp4 del file4

Notemos que magia("archivos/file1.lol") no funcionó, esto nos pareció raro, ya que file1 debería ser el png (siguiendo el hint del pdf de la práctica), pero nuestro método “magia” no lo logro descifrar

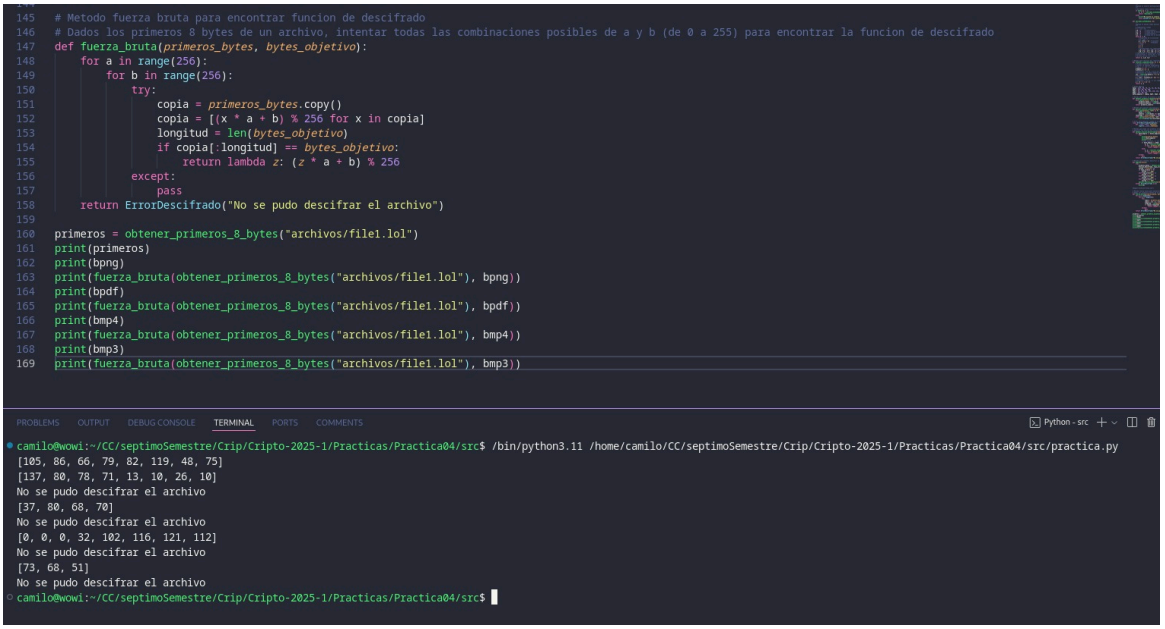
Entonces decidimos intentar la otra manera de descifrar (intentar todas las posibilidades), generar un método para encontrar la función de descifrado para obtener que el inicio sean los bytes mágicos de un tipo de archivo

```python
# Metodo fuerza bruta para encontrar funcion de descifrado
# Dados los primeros 8 bytes de un archivo, intentar todas las combinaciones posibles de a y b (de 0 a 255) para encontrar la funcion de descifrado
def fuerza_bruta(primeros_bytes, bytes_objetivo):
    for a in range(256):
        for b in range(256):
            try:
                copia = primeros_bytes.copy()
                copia = [(x * a + b) % 256 for x in copia]
                longitud = len(bytes_objetivo)
                if copia[:longitud] == bytes_objetivo:
                    return lambda z: (z * a + b) % 256
            except:
                pass
    return ErrorDescifrado("No se pudo descifrar el archivo")
```

Entonces ejecutamos

+ fuerza_bruta(obtener_primeros_8_bytes("archivos/file1.lol"), bpng)
+ fuerza_bruta(obtener_primeros_8_bytes("archivos/file1.lol"), bpdf)
+ fuerza_bruta(obtener_primeros_8_bytes("archivos/file1.lol"), bmp4)
+ fuerza_bruta(obtener_primeros_8_bytes("archivos/file1.lol"), bmp3)

(por si el file1 no era el png) y para nuestra sorpresa nos salió esto



```
# Metodo fuerza bruta para encontrar funcion de descifrado
# Dados los primeros 8 bytes de un archivo, intentar todas las combinaciones posibles de a y b (de 0 a 255) para encontrar la funcion de descifrado
def fuerza_bruta(primeros_bytes, bytes_objetivo):
    for a in range(256):
        for b in range(256):
            try:
                copia = primeros_bytes.copy()
                copia = [(x * a + b) % 256 for x in copia]
                longitud = len(bytes_objetivo)
                if copia[:longitud] == bytes_objetivo:
                    return lambda z: (z * a + b) % 256
            except:
                pass
    return ErrorDescifrado("No se pudo descifrar el archivo")

primeros = obtener_primeros_8_bytes("archivos/file1.lol")
print(primeros)
print(bpng)
print(fuerza_bruta(obtener_primeros_8_bytes("archivos/file1.lol"), bpng))
print(bpdf)
print(fuerza_bruta(obtener_primeros_8_bytes("archivos/file1.lol"), bpdf))
print(bmp4)
print(fuerza_bruta(obtener_primeros_8_bytes("archivos/file1.lol"), bmp4))
print(bmp3)
print(fuerza_bruta(obtener_primeros_8_bytes("archivos/file1.lol"), bmp3))
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1/Practicas/Practica04/src$ /bin/python3.11 /home/camilo/CC/septimoSemestre/Crip/Cripto-2025-1/Practicas/Practica04/src/practica.py
[105, 86, 66, 79, 82, 119, 48, 75]
[137, 80, 78, 71, 13, 10, 26, 10]
No se pudo descifrar el archivo
[37, 80, 68, 70]
No se pudo descifrar el archivo
[0, 0, 0, 32, 102, 116, 121, 112]
No se pudo descifrar el archivo
[73, 68, 51]
No se pudo descifrar el archivo
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1/Practicas/Practica04/src$
```

Lo que significa que file1 no es ningun de esos tipos de archivos, entonces pensamos que seria algun otro tipo de imagen como jpg, pero los jpg empiezan con FF D8 FF pero el file1 empieza con 3 bytes de diferente valor por lo tanto tampoco podría ser. Pero lo que luego notamos (preguntando al ayudante) es que debería estar codificado en base64, entonces el siguiente paso fue hacer un decodificador de base64.
Para esto usamos los siguientes métodos:

+ Método para convertir caracteres de base64 a su valor decimal

```
# Metodo para convertir un caracter de base64 a su valor decimal
def base64_a_decimal(caracter):
    if 'A' <= caracter <= 'Z':
        return ord(caracter) - ord('A')
    if 'a' <= caracter <= 'z':
        return ord(caracter) - ord('a') + 26
    if '0' <= caracter <= '9':
        return ord(caracter) - ord('0') + 52
    if caracter == '+':
        return 62
    if caracter == '/':
        return 63
    raise ValueError("Caracter no valido")
```

+ Método para convertir un numero decimal a su valor binario (usando 6 bits)

```
# Metodo para convertir un numero decimal a su valor binario usando 6 bits
def decimal_a_binario(valor):
    return format(valor, '06b')
```

+ Método para convertir un numero binario a decimal

```
# Metodo para convertir un valor binario a su numero decimal
def binario_a_decimal(valor):
    return int(valor, 2)
```

+ Método para convertir un numero decimal a su carácter en la tabla ASCII (pero de forma hexadecimal)

```python
# Metodo para convertir un numero decimal a su caracter ASCII como hexadecimal
def decimal_a_hexadecimal(valor):
    return format(valor, '02x')
```

Y con esos métodos pudimos crear nuestro método para decodificar texto en base64 y volverlo en una secuencia de bytes (la parte del padding fue algo confusa y tardada, pero con la ayuda de internet y ChatGPT salió)

```python
# Metodo para el algoritmo de descifrado de base64
# Dado un texto cifrado en base64, devolver el texto descifrado como bytes
def decode64(texto):
    # Convertir el texto a una lista de valores decimales, ignorando los caracteres de padding
    valores = [base64_a_decimal(caracter) for caracter in texto if caracter != '=']
    # Convertir los valores decimales a binario
    binario = "".join([decimal_a_binario(valor) for valor in valores])
    # Ajustar la longitud del binario al multiplo de 8 bits (bytes)
    padding = (8 - len(binario) % 8) % 8
    binario = binario[:-padding] if padding else binario
    # Separar el binario en grupos de 8 bits
    grupos = [binario[i:i+8] for i in range(0, len(binario), 8)]
    # Convertir los grupos de 8 bits a su valor decimal
    decimales = [binario_a_decimal(grupo) for grupo in grupos]
    # Convertir los valores decimales a caracteres ASCII como hexadecimal
    hexadecimal = [decimal_a_hexadecimal(decimal) for decimal in decimales]
    # Convertir los caracteres ASCII a bytes
    return bytes.fromhex("".join(hexadecimal))
```

Usando el método de arriba creamos otro método para recibir un archivo con contenido en base64, decodificarlo y guardar el resultado en otro archivo

```python
# Metodo para descifrar un archivo base64 y guardar el resultado
def magia_v2(archivo, archivo_salida):
    with open(archivo, "r") as f:
        texto = f.read()
    datos_descifrados = decode64(texto)
    with open(archivo_salida, "wb") as f:
        f.write(datos_descifrados)
```

De esta forma ejecutamos magia_v2("archivos/file1.lol",”archivos/file.png”) generando un archivo png del file1. De esta manera pudimos recuperar todos los archivos a su forma original.

Ahora veamos la parte extra, para esto tenemos que hacer los cifradores. Empecemos con el más complicado, el cifrador de base64, para este método usamos los siguientes métodos:
Método para convertir un numero decimal a su carácter de base64

```python
# Metodo para convertir un valor decimal a su caracter de base64
def decimal_a_base64(valor):
    if 0 <= valor <= 25:
        return chr(valor + ord('A'))
    if 26 <= valor <= 51:
        return chr(valor - 26 + ord('a'))
    if 52 <= valor <= 61:
        return chr(valor - 52 + ord('0'))
    if valor == 62:
        return '+'
    if valor == 63:
        return '/'
    raise ValueError("Valor no valido")
```

Método para convertir un byte a su valor binario (usando 8 bits)

```python
# Metodo para convertir bytes a su valor binario de 8 bits
def byte_a_binario(byte):
    return format(byte, '08b')
```

Método para convertir un numero binario a su valor decimal

```python
# Metodo para convertir un binario a su valor decimal
def binario_a_byte(binario):
    return int(binario, 2)
```

Y con esos métodos pudimos crear nuestro método para codificar una secuencia de bytes y volverlo en texto en base64 (la parte del padding fue algo confusa y tardada, pero la ayuda de internet y ChatGPT salió)

```python
# Metodo para el algoritmo de cifrado de base64
# Dado un texto en bytes, devolver el texto cifrado en base64
def encode64(bytes):
    # Convertir los bytes a valores binarios
    binarios = [byte_a_binario(byte) for byte in bytes]
    # Unir los valores binarios en un solo string
    binario = "".join(binarios)
    # Separar el binario en grupos de 6 bits
    grupos = [binario[i:i+6] for i in range(0, len(binario), 6)]
    # Si el ultimo grupo tiene menos de 6 bits, agregar ceros al final y calcular el padding
    if len(grupos[-1]) < 6:
        padding = 6 - len(grupos[-1])
        grupos[-1] += "0" * (6 - len(grupos[-1]))
    else:
        padding = 0
    # Convertir los grupos de 6 bits a su valor decimal
    decimales = [binario_a_byte(grupo) for grupo in grupos]
    # Convertir los valores decimales a caracteres de base64
    base64 = [decimal_a_base64(decimal) for decimal in decimales]
    # Calcular el padding en base a los bits faltantes
    padding = "=" * (padding // 2)
    # Regresar el resultado como texto base64 con el padding si es necesario
    return "".join(base64) + padding
```

Usando el método de arriba creamos otro método para recibir un archivo, codificar y guardar el resultado en otro archivo

```python
# Metodo para cifrar un archivo y guardar el resultado
def magiant_v2(archivo, archivo_salida):
    with open(archivo, "rb") as f:
        datos = f.read()
    texto_cifrado = encode64(datos)
    with open(archivo_salida, "w") as f:
        f.write(texto_cifrado)
```

Para probar nuestro codificador de base64 hicimos las siguientes pruebas:

+ magiant_v2("archivos/file1.png", "archivos/file1.xd")
+ magia_v2("archivos/file1.xd", "archivos/file5.png")
+ magiant_v2("archivos/file2.pdf", "archivos/file2.xd")
+ magia_v2("archivos/file2.xd", "archivos/file6.pdf")
+ magiant_v2("archivos/file3.mp3", "archivos/file3.xd")
+ magia_v2("archivos/file3.xd", "archivos/file7.mp3")
+ magiant_v2("archivos/file4.mp4", "archivos/file4.xd")
+ magia_v2("archivos/file4.xd", "archivos/file8.mp4")

Y por último para el cifrado de archivos usando una función afín usamos los métodos que ya teníamos, creando un método que recibe un archivo, se cifra usando los valores dados y lo guarda

```python
# Metodo para cifrar un archivo usando una funcion afin y guardar el resultado en un nuevo archivo
def magiant(archivo, a, b, archivo_salida):
    funcion_cifrado = lambda z: (a * z + b) % 256
    aplicar_funcion_a_bytes(archivo, funcion_cifrado, archivo_salida)
```

Para probar nuestro cifrador hicimos las siguientes pruebas:

+ magiant("archivos/file1.png", 3, 5, "archivos/file11.owo")
+ magia("archivos/file11.owo")
+ magiant("archivos/file2.pdf", 11, 17, "archivos/file12.owo")
+ magia("archivos/file12.owo")
+ magiant("archivos/file3.mp3", 13, 7, "archivos/file13.owo")
+ magia("archivos/file13.owo")
+ magiant("archivos/file4.mp4", 19, 4, "archivos/file14.owo")
+ magia("archivos/file14.owo")

Preguntas:

1. **¿Cuántos primos relativos hay en Z_{256}?**

   <u>En total hay 128 primos relativos en Z_{256}</u>**:**

   Esto lo podemos calcular con la función φ, de la siguiente forma:

   Sabemos que 256=2^8 y por propiedad de la función φ sabemos que si n es un número primo y k es un número natural entonces φ(n^k)=n^k-n^{k-1}, por lo que:

   φ(256)=φ(2^8)=2^8-2^7=256-128=128

2. **Sea f :$Z_n \to Z_n$ tal que $f(i) = k * i$ para i ∈ A, con A un alfabeto cualquiera.**

   ¿Qué se debe cumplir para que f sea una función biyectiva?

   Nos dicen que $f(i) = k * i$ es una función en f:$Z_n \to Z_n$, y segun el Blog de Leo una función es biyectiva si es **inyectiva** y **suprayectiva**, esto quiere decir que:

   + **Inyectiva:** manda distintos elementos del dominio a distintos elementos del contradominio
   + **Suprayectiva:** todo el contradominio tiene su correspondencia.

   En pocas palabras podrán “trasladar” todo un conjunto a otro [17]. También según el Blog de Leo para una función $f\colon X \to Y$ las siguiente tres cosas son equivalentes:

   + f es biyectiva
   + f tiene inversa
   + f tiene inversa derecha y f tiene inversa izquierda [18]

   Para este caso nos interesa saber si $f(i) = k * i$ tiene inversa, si $f(i) = k * i$ tiene inversa entonces es una función biyectiva. Para esto necesitamos que k tenga inverso multiplicativo en Z_n y esto pasa unicamente si k es coprimo con n, i.e. que el máximo común divisor de ambos números sea uno $mcd(k, n) = 1$. Y obviamente k debe ser distinto de 0, si no todo lo mandaría al 0😆.

3. **¿Cuántas posibles combinaciones no triviales existen para cifrar bytes con César, Decimado y Afín?**
   - **César:**
     Observamos que en el cifrado césar en bytes tiene un total de 255 posibles cifrados no triviales, debido a que es un trivial el que consiste de rotar 0 espacios, dejándonos 255 posibles corrimientos y por ende 255 posibles combinaciones no triviales.
   - Decimado:
     Para que el número a multiplicar nos de un alfabeto válido este debe de ser un primo relativo con 256. Previamente habíamos visto que en total 256 tiene 128 primos relativos, por lo cual vamos a tener 128 posibles combinaciones válidas y no triviales.
   - Afin:

Sabemos que los cifrados afines son de la siguiente forma:

$$f(x)=(ax+b) mod n$$

Donde x representa el valor numérico del símbolo a cifrar, b es cualquier valor numérico de 0 a n–1 y a es primo relativo con n, siendo n el tamaño del alfabeto a cifrar.

Observemos que el total de cifrados posibles va a ser de el valor de todas las posibles a por todas las posibles b.
En nuestro caso, previamente obtuvimos que 256 tiene un total de 128 primos relativos por lo cual tenemos 128 probabilidades para a. b puede tomar un valor de 0 a 255 lo cual nos da una probabilidad de 256 valores distintos. Dándonos en total 32768 posibles combinaciones válidas únicas. Ahora, observemos que si a=1 y b= 0 estaremos cifrando cada una de los símbolos del alfabeto con sí mismos, volviendolo trivial. Debido a lo anterior vamos a tener un total de 32767 posibles cifrados no triviales.

4. **¿Por qué el sistema de archivos de UNIX, aunque un archivo tenga una extensión diferente (o incluso no tenga), sigue reconociendo al archivo original?**
   En el sistema de archivos de UNIX no se fia de la extensión para reconocer el tipo de archivo, por lo que no tienen por qué tener extensiones finales como en otros sistemas operativos. Sin embargo, hay ciertas aplicaciones que si requieren la extensión de los archivos, como por ejemplo Adobe's Acrobat Reader [14]. Además linux cuenta con el el comando **file** para organizar la información de manera rápida, este comando es utilizado para determinar el tipo de archivo ejecutando 3 pruebas:
   + **Filesystem text:** examina si es un archivo vacío o un tipo de archivo especial, además de verificar si es un formato conocido que sea relevante para el sistema.
   + **Magic Test:** utiliza "numeros mágicos", una cadena córta de números al inicio del archivo para revisar si se trata de datos binarios ejecutables
   + **Language test:** examina el conjunto de caracteres en los que está escrito el archivo, por ejemplo ASCII o UTF-8. Esta prueba se realiza al final por ser menos precisa.

   De esta forma, el sistema de archivos de UNIX puede reconocer el tipo de archivo sin importar si tiene una extensión distinta o no cuente con una [15].
5. **¿Por qué los archivos descifrados tienen exactamente el mismo tamaño que antes de cifrar, pero no pudimos leerlos? ¿Por qué no tuvimos que agregar/quitar nada?**
   Esto se debe a que al cifrar mediante Cesar, diezmado o afín, lo único que estamos haciendo es sustituir valores por otros manteniendo la cantidad de elementos. Es decir, como solo estamos reemplazando valores por otros se conserva el tamaño original, pero no podemos leerlos ya que que estos nuevos símbolos no tienen el mismo significado ni valor que los originales por lo cual aunque sea el mismo tamaño no tienen el mismo significado. No tuvimos que agregar/quitar algo, por lo mismo. Solo estamos cambiando símbolos del texto plano por otros símbolos ya sea del mismo alfabeto u otro, siempre que se mantenga el número de caracteres.

6. **Ya que base64 no es un cifrado, sino codificación, ¿en qué casos podemos usarlo?**
   Base64 es un sistema de codificación que se utiliza para representar datos binarios (como archivos, imágenes, documentos, etc.) en una forma de texto ASCII . La razón principal de su uso es permitir la transferencia y almacenamiento de datos binarios a través de medios que podrían no ser capaces de manejar datos binarios directamente, como enlaces de correo electrónico, URL, sistemas de transmisión de texto plano, entre otros. Al convertir los datos binarios en una cadena de caracteres ASCII, se evitan problemas de interpretación y corrupción que podrían surgir al transmitir datos binarios a través de estos medios [16]. Personalmente entendí esto mejor cuando implementé el backend de una aplicación de e-commerse, tuve que encontrar una manera de envíar las imagenes al frontend para poder mostrarlas en la página. La solución fue enviar imagenes codificadas en base64.

7. **Supongamos que estuvieras en Hogwarts y tuvieras que utilizar un búho para comunicarte, ¿cuál crees que sería la mejor opción para mandar mensajes seguros a través de la lechuza? Ya que podría caer en manos equivocadas. ¿Cómo cifrarías tu mensaje, o cómo harías el intercambio de llaves? Puedes darte una idea con este video. Puedes ser creativo :)**

   Supongamos que soy Harry y quiero mandar un mensaje a Sirius Black, primero escribo una carta con el mensaje que quiero transmitir, después aplico un hechizo que solo yo conozco a la carta, a partir del hechizo que solo yo conozco genero otro hechizo para aplicarlo a la carta y poder abrirla, este hechizo lo escribo y lo envío a Sirius, en vez de enviar mi propia lechuza utilizo las lechuzas de la escuela ya que se suponen que es un medio confiable, además al enviarla a través de una lechuza de la escuela, si un mortífago quiere conocer los mensajes que le envío a Sirius tendría que interceptar todas o una gran parte de las lechuzas de la escuela para obtener el hechizo que generé a partir del hechizo que solo yo conozco. Después aplico un segundo hechizo que solo yo puedo deshacer a una parte de la carta, cargo a mi lechuza con la carta y una pregunta cuya respuesta solo la conocemos Sirius y yo, antes de envíar a mi lechuza le aplico un hechizo de invisibilidad para que sea más complicado de seguir e interceptar.

   Una vez Sirius reciba la carta de mi lechuza personal y el hechizo por parte de una lechuza de la escuela, aplica un hechizo que solo el puede deshacer a la otra parte de la carta que no tiene mi hechizo aplicado, después escribe la respuesta a la pregunta, aplica el hechizo de invisibilidad y envía a la lechuza de regreso a mí.

   Una vez mi lechuza regresa, verificó que la respuesta a la pregunta sea la correcta, después deshago el segundo hechizo que le aplique una parte de la carta y envío la confirmación de la respuesta a la pregunta con el hechizo que solo puede deshacer el hechizo que le envíe a Sirius Black. Vuelvo a aplicar el hechizo de invisibilidad y regreso la lechuza con la carta y la confirmación de la pregunta hechizada.

Una vez Sirius recibe la lechuza, utiliza el hechizo que le envíe para deshacer el hechizo de la confirmación de la respuesta, verifica que la confirmación sea positiva, deshace el hechizo que aplico a la otra parte de la carta y finalmente utiliza el hechizo que le envíe para deshacer el hechizo y poder leer el mensaje.

Resumen: para esta idea se decidió combinar el protocolo Three Pass Protocol (visto en el video anexado a esta tarea) en combinación con un cifrado asimétrico (como el protocolo SSH que vimos en la primera práctica de laboratorio) con una segunda capa de seguridad que consiste en verificación (en este caso la pregunta de seguridad) y mecanismos de seguridad como hechizos de invisibilidad y medios seguros de comunicación como lo son el sistema de mensajería de Búhos por parte de Hogwarts.

Conclusiones:

Consideramos que es mala idea usar cualquiera de estos cifrados, afuera del estudio teórico o que deliberadamente quieras que los rompan, debido a que, como vimos previamente en conjunto tienen un número bastante pequeño de posibles cifrados y con el avance de las computadoras y el enorme número de operaciones que estas pueden hacer por segundo hacen que romper el cifrado a fuerza bruta o por congruencias sea bastante sencillo.
En esta práctica aprendimos como realizar un decodificador de base 64 sin el uso de bibliotecas especializadas, al igual que el nombre del cifrado decimado el cual previamente conocíamos por otro nombre y como este con el de César forman los cifrados afines.

Bibliografía:

1. *Función \(\varphi\) de Euler y Teorema de Euler - Fermat*. (s. f.). https://www.matematicas.ciencias.uchile.cl/juaco/section-15.html#:~:text=nk%E2%88%921.-,%CF%86%20(%20n%20k%20)%20%3D%20n,k%20%E2%88%92%20n%20k%20%E2%88%92%201%20.
2. Antonio Cedillo Hernandez. (2020, October 5). Seguridad Informática I - Inverso Modular (Cifrado Asimétrico RSA). YouTube. https://www.youtube.com/watch?v=B6FuVRPRT4Y
3. lasmatematicas.es. (2011, February 27). Reducir módulo un número - división. YouTube. https://www.youtube.com/watch?v=MGGm4ZIcgkE
4. lasmatematicas.es. (2021, December 14). ✅ MINICURSO sobre ARITMÉTICA MODULAR 😍😍. YouTube. https://www.youtube.com/watch?v=OZLm2yl-oqs
5. de, C. (2008, September 10). Inverso multiplicativo (aritmética modular). Wikipedia.org; Wikimedia Foundation, Inc. https://es.wikipedia.org/wiki/Inverso_multiplicativo_(aritm%C3%A9tica_modular)
6. Wikipedia Contributors. (2019, October 19). List of file signatures. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/List_of_file_signatures
7. *¿Cuál es el espacio clave de un cifrado afín?* (2024, 9 agosto). es.eitca.org. https://es.eitca.org/cybersecurity/eitc-is-ccf-classical-cryptography-fundament

als/history-of-cryptography/modular-arithmetic-and-historical-ciphers/what-is-the-key-space-of-an-affine-cipher/
8. *Cifrado afín - Wikiwand articles. (s. f.).* *https://www.wikiwand.com/es/articles/Cifrado_af%C3%ADn#google_vignette*
9. Bruff, D. (2017, 6 septiembre). Decimation Ciphers – Cryptography. https://derekbruff.org/blogs/fywscrypto/2017/09/06/decimation-ciphers/
10. Base64 Decode Algorithm | Base64 Algorithm | Learn | Base64. (n.d.). Base64.Guru. https://base64.guru/learn/base64-algorithm/decode
11. Base64 Encode Algorithm | Base64 Algorithm | Learn | Base64. (n.d.). Base64.Guru. https://base64.guru/learn/base64-algorithm/encode
12. Hemant. (2024). Why padding is used in Base64 encoding? Stack Overflow. https://stackoverflow.com/questions/4322507/why-padding-is-used-in-base64-encoding
13. Wikipedia Contributors. (2019, November 10). Base64. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Base64
14. Mark A. Thomas. (2019). The Unix File System https://homepages.uc.edu/~thomam/Intro_Unix_Text/File_System.html
15. Edward S. y Noviantika G. (2023, Diciembre 22). The Linux File Command: How to Use It to Determine a File Type https://www.hostinger.com/tutorials/linux-file-command/
16. Arantxa Abad (s.f.) Base64: ¿Qué es y para qué se usa? https://modocreativo.es/que-es-base64/
17. Guillermo Oswaldo Cota Martínez. (s.f.). Álgebra Superior I: Funciones inyectivas, suprayectivas y biyectivas https://blog.nekomath.com/algebra-superior-i-funciones-inyectivas-suprayectivas-y-biyectivas/
18. Gabriela Hernández Aguilar. (s.f.). Teoría de los Conjuntos I: Funciones inversas https://blog.nekomath.com/teoria-de-los-conjuntos-i-funciones-inversas/