UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

Guatemala 28 de enero de 2025

Jose Santisteban - 21153

Cifrado de información

Ejercicio 1

Cifrado elegido: Afín

Una de las partes más importantes de este cifrado es el cálculo del máximo común divisor, ya que determina si el valor de a, uno de los parámetros del cifrado, es válido. En este cifrado, el valor de aaa debe ser coprimo con el tamaño del alfabeto (en este caso, 26).

```python
def mcd(a, b):

    while b:

        a, b = b, a % b

    return a
```

Otro elemento importante del cifrado afín, es el inverso modular. Para descifrar un texto cifrado, necesitamos invertir la operación de cifrado. Esto implica encontrar un número x tal que $(a \cdot x) \bmod m = 1$. Este valor es el inverso modular de a respecto a m. Si a y m no son coprimos, este inverso no existirá, lo que refuerza la importancia de la verificación del MCD.

```python
def inverso_modular(a, m):

    for i in range(1, m):

        if (a * i) % m == 1:

            return i

    raise ValueError(f"No existe inverso modular para a={a} y m={m}")
```

La función principal del cifrado afín toma un texto, un valor a y un valor b (el desplazamiento). Primero, verifica si aaa es válido calculando el MCD entre aaa y 26. Luego, aplica la fórmula matemática para cifrar cada letra: $C(x)=(a \cdot x+b) \bmod 26$. Aquí, x es la posición de la letra en el alfabeto (por ejemplo, A=0,B=1,...Z=25). Después, convierte la posición cifrada nuevamente a un carácter. Los caracteres no alfabéticos, como espacios o números, no se modifican y se agregan al texto cifrado tal cual.

```
def cifrar_afin(texto, a, b):

    if mcd(a, 26) != 1:

        raise ValueError("El valor de 'a' debe ser coprimo con 26.")

    texto_cifrado = ""

    for char in texto:

        if char.isalpha():  # Solo cifrar letras

            base = ord('A') if char.isupper() else ord('a')

            x = ord(char) - base

            # Aplicar fórmula del cifrado afín

            cifrado = (a * x + b) % 26

            texto_cifrado += chr(cifrado + base)

        else:

            texto_cifrado += char  # Mantener caracteres no alfabéticos

    return texto_cifrado
```

En el caso del descifrado, se tiene que hacer la operación inversa al cifrado. En este caso se necesitan los valores de a y b del cifrado así como el inverso modular de a.El descifrado se realiza recorriendo cada carácter del texto cifrado. Si el carácter es una letra, se convierte a su posición numérica dentro del alfabeto, utilizando ord('A') como base para mayúsculas y ord('a') para minúsculas. Luego se aplica la fórmula para el descifrado:

$$D(a, b, \mathbf{x}) = a^{-1}(\mathbf{x} - b) \pmod{M}$$

Dando como resultado la letra original del texto cifrado.

```
def descifrar_afin(texto_cifrado, a, b):
    if mcd(a, 26) != 1:
        raise ValueError("El valor de 'a' debe ser coprimo con 26.")

    texto_descifrado = ""
    a_inv = inverso_modular(a, 26)  # Obtener el inverso modular de 'a'
    for char in texto_cifrado:
        if char.isalpha():  # Solo descifrar letras
            base = ord('A') if char.isupper() else ord('a')
            x = ord(char) - base
            # Aplicar fórmula del descifrado afín
            descifrado = (a_inv * (x - b)) % 26
            texto_descifrado += chr(descifrado + base)
        else:
                texto_descifrado += char   # Mantener caracteres no
alfabéticos
    return texto_descifrado
```

Se eligió el cifrado afín debido a que combina dos tipo de transformaciones básicas, las cuales son multiplicación modular y suma modular, lo que representa un nivel de complejidad más alta a comparación del cifrado césar el cual solo consiste en desplazamiento básico. Además de que se requieren los valores de a y b para poder hacer el descifrado, lo que hace más robusto el algoritmo.

Las ventajas del algoritmo afín erradican en la simplicidad en su implementación como en la aplicación ya que solo requiere operaciones básicas de aritmética, lo que lo hace accesible a personas sin conocimiento avanzado de matemáticas. Además por la naturaleza del algoritmo es posible generar una mayor cantidad de claves a comparación de cifrados más simples, lo que aumenta su resistencia frente a ataques de fuerza bruta.

A pesar de ser un cifrado robusto para su época, este cifrado presenta vulnerabilidades claras para tiempos actuales, ya que es susceptible a un análisis de frecuencia ya que cada letra en el texto original se mapea en el texto cifrado, manteniendo los patrones de idioma.