UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ingeniería

Ingeniería en Computación

CRIPTOGRAFÍA (2010)

PROYECTO 1

Grupo: 01

Profesora: Ing. Magdalena Reyes Granados

Equipo 3

Salinas Gutiérrez Gerardo

Valeriano Barrios Cristian

Semestre 2022-1

Fecha: 22/10/2021

INTRODUCCIÓN

La criptografía es un elemento indispensable en los sistemas de información, no solo para cumplir estándares sino además para cumplir con requerimientos legales. Por ello, es importante que como profesionistas de la computación comprendamos el funcionamiento de los mecanismos de cifrado y además seamos capaces de implementarlos en un escenario real.

Cifrado Transposición por Columnas:

En este tipo de cifrado, unidades de texto plano se cambian de posición siguiendo un esquema bien definido; las “unidades de texto” pueden ser de una sola letra (el caso más común), pares de letras, tríos de letras, mezclas de lo anterior. Este tipo de cifradores eran muy usados en la criptografía clásica y por tanto, al tener que hacer los cálculos por medios muy básicos, normalmente el algoritmo se basaba en un diseño geométrico o en el uso de artilugios mecánicos. Este tipo de algoritmos son de clave simétrica porque es necesario que tanto el que cifra como el que descifra sepan la misma clave para realizar su función.

Cifrado Afín:

El cifrado afín también se le llama cifrado de transformación afín o cifrado monoalfabético genérico. Es un tipo de cifrado por sustitución en el que cada símbolo del alfabeto en claro (el alfabeto del texto en claro) es sustituido por un símbolo del alfabeto cifrado (el alfabeto del texto cifrado) siendo el número de símbolos del alfabeto en claro igual que el número de símbolos del alfabeto cifrado. Para hallar el símbolo del alfabeto cifrado que sustituye a un determinado símbolo del alfabeto en claro, se usa una función matemática afín en aritmética modular.

$$c_i = (a * m_i + b) \mod n$$

Donde:

c_i : Identifica el símbolo i del texto cifrado

a: Se la llama constante de decimación

m_i : Identifica el símbolo i del texto en claro

b: Se la llama constante de desplazamiento

n: Es el número de símbolos del alfabeto de cifrado (el orden)

Cifrado Vernam:

El cifrado vernam está definido por el alfabeto A = {0,1}, un mensaje binario *m1,m2,...,mt* como operando de una clave binaria *k1,k1,...,kt* de la misma longitud, produciendo una cadena cifrada *c1,c2,...,ct*, donde:

$$ci = mi \oplus ki,\ 1 \leq i \leq t.$$

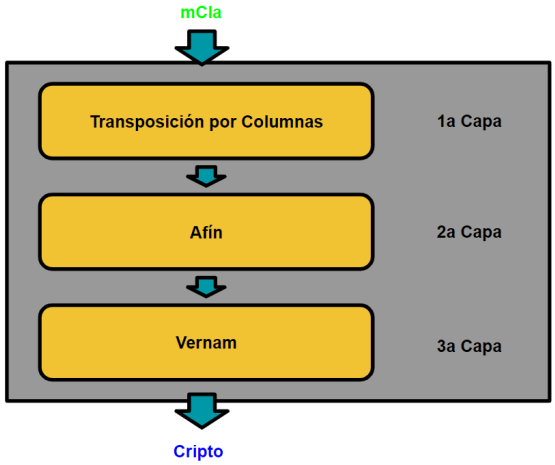
La ventaja de este cifrado es que el mensaje cifrado resulta completamente ilegible, ya que la operación XOR entre el mensaje y la clave, genera caracteres ASCII, por lo que de no contar con la clave exacta, sería muy difícil descifrarlo.

JUSTIFICACIÓN

Los algoritmos de cifrado solicitados en este trabajo son Vernam, Transposición por columnas y Afín.

Sin embargo, por la limitación de utilizar alfabetos de 26 o 27 caracteres para el cifrado Afín, resultaría complicado trabajar con un mensaje que previamente ha pasado por un cifrado Vernam, el cúal da como resultado un mensaje con un alfabeto de 128 caracteres ASCII.

Es por ello que se ha modificado el orden de las capas de cifrado, quedando como resultado:



El descifrado se realiza ejecutando las capas en el orden inverso.

También es importante destacar que el mensaje al ser leído, por simplicidad, elimina los espacios y convierte el texto a mayúsculas.

DESARROLLO

Inicialmente se pensó en utilizar C++, pero al cabo de unos intentos nos dimos cuenta de que las líneas de código eran demasiadas y la depuración sería más complicada.

Por ello, decidimos desarrollar el cifrador en el lenguaje Python (versión 3) debido a que ofrece bastantes funciones que nos simplificarían el proceso y el número de líneas disminuyó bastante.

Primera capa: Transposición por Columnas

Como primera etapa, se encarga de leer el archivo de texto con nuestro mensaje a cifrar (Equipo3.txt), en él como en los algoritmos posteriores se convierte el texto a mayúsculas y se eliminan los espacios.

```
f = open("mensaje.txt","r")
if f.mode == "r":
    mensaje = f.read() #se guarda el mensaje leído
print("\nMensaje leído:", mensaje.upper())
f.close() #cierre del archivo
```

Acto seguido, el programa pedirá ingresar una clave la cual para nuestro ejercicio, no deberá contar con caracteres repetidos. La clave será tratada como el mensaje

```
mensaje = mensaje.replace(' ', '').upper() #se elimina cualquier espacio y se convierte en mayúsculas

#ingresamos la clave, esta debe tener caracteres NO repetidos
clave = str(input("\nIntroduce la clave sin repetir letras: "))
clave = clave.replace(' ', '').upper() #se elimina cualquier espacio y se convierte en mayúsculas
```

El proceso de cifrado consiste, a grandes rasgos, en calcular la matriz de cifrado basado en la longitud de nuestro mensaje original y hacer el ordenamiento correspondiente. Con base en la posición de cada índice se hará un ordenamiento alfabético y de esta manera, se conseguirá el cifrado. El proceso de descifrado es bastante similar, partiendo de la matriz, ahora de descifrado, se hará una lectura y los caracteres se organizarán de manera que puedan ser legibles.

```
print("\n--------------------",
      "\nProceso de descifrado",
      "\n--------------------")

msgLength = len(mcifrado) #tamaño del mensaje a descifrar
msgList = list(mcifrado) #el mensaje se divide en elementos de una ['L', 'I', 'S', 'T', 'A']

columnas = len(clave) #se determina el número de columnas según el tamaño de la clave
filas = int(math.ceil(msgLength/columnas)) #se calculan las filas, math.ceil devuelve el entero mayor o más próximo de la división del msgLength/columnas

descifrado = [] #se crea la variable de la matriz de descifrado

for i in range(filas):
    descifrado += [[''] * columnas] #se crea la matriz de descifrado

claveApuntador = 0 #variable del apuntador de la clave
mensajeApuntador = 0 #variable del apuntador del mensaje
claveOrdenada = sorted(list(clave)) #la clave se divide en elementos de una lista y se ordena alfabéticamente
```

Segunda capa: Cifrado Afín

En esta etapa se recibirá la cadena de caracteres tratada previamente por el algoritmo de Transposición por Columnas, pide ingresar las constantes de decimación y de desplazamiento (para el cifrado y el descifrado).

```
# C(i)=(a*Mi+b)mod(n)

print("\n------------------",
      "\nProceso de cifrado",
      "\n------------------")

#condición para la constante de desplazamiento
if b>n or b<0:
    b=b%n

#lectura letra por letra del mensaje
for i in mclaro:
    #alfabeto.find(i) por cada caracter leído, encuentra su posición dentro del alfabeto
    x=a*alfabeto.find(i)+b #se realiza la operación (a*Mi+b)
    cifrado+=alfabeto[x%n] #se aplica el mod(n) dentro de alfabeto[] para encontrar el caracter según la posición

#archivo abierto en modo escritura
f = open("mensajecifrado.txt","w", encoding='utf-8')
if f.mode == "w":
    f.write(cifrado) #se guarda el mensaje cifrado
f.close() #cierre del archivo
```

En el descifrado, el programa pedirá se ingrese de nuevo las constantes, se encargará de encontrar la inversa y por aritmética modular obtendrá el mensaje descifrado.

```
for a1 in range(0,n): # mínimo común divisor
    if (a1*a)%n==1:
        break

print("El valor de a =", a, "el valor de a^-1 =", a1)

#lectura letra por letra del mensaje
for i in mcifrado:
    #alfabeto.find(i) por cada caracter leído, encuentra su posición dentro del alfabeto
    x=a1*(alfabeto.find(i)-b) #se realiza la operación (a*Mi-b)
    descifrado+=alfabeto[x%n] #se aplica el mod(n) dentro de alfabeto[] para encontrar el caracter según la posición

print("\n---------------------",
      "\nProceso de descifrado",
      "\n---------------------")

#archivo abierto en modo escritura
f = open("mensajecifrado.txt","w")
if f.mode == "w":
    f.write(descifrado) #se guarda el mensaje cifrado
f.close() #cierre del archivo
```

Tercera capa: Cifrado Vernam

En esta etapa se utiliza un mensaje que ya tiene un alto nivel de encriptamiento.

Primero se utiliza esta función básica para leer el mensaje contenido en un archivo de texto llamado ‘mensajecifrado.txt” que tiene que estar contenido en el mismo

directorio donde se ejecuta el programa, y se pasa a la variable local “mCla” que es la cadena que comenzaremos a manipular:

```
f = open("mensajecifrado.txt","r")
if f.mode == "r":
    mCla = f.read() #se guarda el mensaje leído
print("\nMensaje leído:", mCla)
f.close() #cierre del archivo
```

Posteriormente se calcula la longitud de la cadena mCla para poder generar una clave aleatoria única, de la misma longitud, la cuál se guarda en la variable “key”:

```
###     Generando clave única
n = len(mCla)   #Longitud del mCla para generar una máscara de la misma longitud
key = ''.join(random.choice(string.ascii_uppercase + string.digits) for i in range(n))  #Genera una clave aleatória
```

Finalmente, con la siguiente función se realiza la operación XOR caracter a caracter y el resultado se guarda en la variable “cifrado”:

```
#Proceso de cifrado = (mCla) XOR (key)
for i, c in enumerate(mensajeStr, start=0):     #Recorre el mensaje caracter a caracter
        aux_xor = xor(ord(mensajeStr[i]), ord(keyStr[i]))       #Operación XOR entre cada par de caracteres
        cifrado += chr(aux_xor) #Guardando el resultado de la operación en la cadena "cifrado"
```

Tanto la clave como el resultado se guardan en archivos de texto “clave.txt” y “mensajecifrado.txt” respectivamente.

Ambos archivos son necesarios posteriormente para realizar el descifrado realizando de nuevo una operación XOR entre la clave y el criptograma.

CONCLUSIONES

Este proyecto nos ayudó a reforzar nuestros conocimientos de la materia, pero también para reforzar nuestras habilidades de programación, ya que implicó cierto nivel de complejidad sincronizar los algoritmos individuales, y requirió segmentar el cifrador en distintos módulos para hacerlo más óptimo y fácil de editar.

Sin embargo, hubo complicaciones en la etapa de cifrado Vernam, algunos caracteres se modificaron por las conversiones entre string estándar de Python y las funciones de conversión a ASCII. Con más tiempo es un problema que se puede solucionar y una oportunidad para desarrollar un programa más estable, robusto y complejo.

REFERENCIAS

MENEZEZ A.. (2001). Handbook of applied criptography. Boca Raton, FL: CRC Press.