CIFRA DE AFÍN, SUSTITUCIÓN SIMPLE Y SU ATAQUE BELLASO Y VIGENERE COMO CIFRAS POLIALFABÉTICAS

AUTORA: HELLEN CUBERO LEDEZMA

OCTUBRE: 2020

San Marcos

Introducción

Para aquellos estudiosos de la ingeniería en sistemas y amantes de la criptografía vista desde Python, el poder estudiar temas como los que en este resumen se abordan, resultará una gran herramienta al momento de enriquecer su conocimiento.

Es por esto que, en las siguientes páginas se muestra de manera profunda, métodos y técnicas de cifrado, descifrado monoalfabéticos y polialfabéticos utilizados desde Python. El lector conocerá, aspectos como breves reseñas de las diferentes técnicas de cifrado, sus diferencias y similitudes representadas de manera implícita, así como el código fuente, y la implementación del mismo.

Se debe aclarar que lo que aquí se expone es un resumen de los capítulos 10, 11, 12, 13, 14 y 15 del libro “criptografía sin secretos con Python de David Arboledas Brihuega del 2017, por lo que, sí no se logra comprender algún punto, en varias secciones de la lectura se recomienda acudir a dicho documento.

Contenido

La cifra Afín

Este documento se encuentra basado en su totalidad en los capítulos **10, 11, 12, 13, 14 y 15** del libro **criptografía sin secretos con Python** de David Arboledas Brihuega del 2017

Según Arboledas 2017 esta cifra es un tipo de cifrado por sustitución monoalfabética en el que cada símbolo del alfabeto en claro se sustituye por otro símbolo del alfabeto cifrado, de modo que el número de caracteres de ambos alfabetos sean iguales. El nombre de este algoritmo se debe a que para encontrar que símbolo del alfabeto cifrado sustituye a un determinado símbolo del alfabeto en claro, se usa una **función matemática afín** en **aritmética modular.** (197)

La aritmética puede ser elaborada de manera matemática con la relación de congruencia entre número enteros. Entiéndase **congruencia** cuando dos números enteros tienen el mismo resto al momento de dividirlos por número natural, esto se llama módulo. Se representa así: ***a=b (mod m). La letra m es el módulo de congruencia.***

Visualiza el módulo con relojes

Número	0	1	2	3	4	5	6
Módulo	0	1	2	0	1	2	0

Fuente: Arboledas, 2017

En la imagen anterior se observa como cuando el número incrementa de uno en uno y cuando se divide en 3 el módulo cambia.

El resto de la división empieza en 0 y aumenta de uno en uno hasta que llega al número entre el cual estamos dividiendo, en el que vuelve a ser cero, luego de esto la secuencia se repite.

Para encontrar el operador módulo usando relojes, se utilizará la llamada aritmética del reloj (Hoffstein, 2008) que consiste en que se escribe el 0 a las doce en punto y se continúa escribiendo números enteros comenzando con 1 hasta el valor m-1 en dirección de las manecillas del reloj.

Ejemplo: Para encontrar el resultado *a mod m,* se harán los siguientes pasos:

Paso 1: dibujar un reloj tamaño m.

Paso 2: empezar en 0 y continúa en sentido de las agujas del reloj a pasos. Donde quede, ese es el resultado. Si el número es positivo se avanza en la forma de horario normal, de lo contrario se retrocede de forma antihoraria.

El operado de módulo en Python

Python utiliza el símbolo de porcentaje (%) para referirse al operador módulo.

```
>>> 5 % 3  # Se lee cinco módulo 3
2
>>> (5 + 3) % 3
2
>>> (5 + 2 * 3) % 3
2
>>>
```

Se puede pensar en el módulo como el resto de la división, entonces se vería así, 29 / 7 = 4 resto 1 y por eso 29 % 7 = 1. Esto funciona para los números positivos, sin embargo, no sirve siempre para los negativos. Por ejemplo: -29 / 7 = -4 resto -1, pero el resultado de una operación módulo no puede ser nunca negativo. El resto -1 es lo mismo que 7 -1, que es 6. Por ello -29 % 7 es 6:

```
>>> - 29 % 7
6
>>>
```

Operaciones en la cifra afín

Como ya se sabe, se debe usar una función matemática en aritmética modular para hallar el símbolo del alfabeto cifrado que sustituye a un determinado símbolo del alfabeto en claro con la cifra afín. Para ello se necesita asignar un orden a cada símbolo de cada uno de los alfabetos.

Por ejemplo: se emplea un alfabeto de 26 caracteres, entonces los símbolos abarcaran desde el 0 (A) hasta el 25 (Z). Una vez establecido el orden, la fórmula para cifrar cualquier símbolo del texto plano tendrá la siguiente forma:

$$c_i = (a \bullet m_i + b) \bmod n$$

c_i: i del texto cifrado.
a: es la llamada constante de decimación.
m_i: i del texto en claro.
b: se denomina constante de desplazamiento.
n: es el número de símbolos del alfabeto de cifrado.

Los sistemas de cifrado afín se pueden definir según el valor de los valores a y b:

Si a= 1 se dice que el sistema es un cifrado por desplazamiento puro.
Si b = 0 se dice que el cifrado es por decimación pura.
Si $a \neq 1$ y b ≠ 0, se dice que el sistema es un cifrado por sustitución afín.

Para descifrar un símbolo C_i habrá que realizar el proceso inverso, que se puede describir con la función matemática:

$m_i. = a^{-1}(c_i - b) \bmod n$

Donde a^{-1} representa el inverso modular o multiplicativo, es decir, el menor número entero que cumpla que $a - a^{-1} \bmod n = 1$. Para que esto ocurra es necesario que a y n sean primos entre si.

Máximo común divisor. Algoritmo de Euclides*

El máximo común divisor (**mcd**) de dos números enteros es el mayor de los divisores comunes de ambos números sin dejar resto. Un método para calcular el **mcd** de dos números es el algoritmo de Euclides, en el que se emplea el algoritmo de la división junto al hecho de que el **mcd** también divide al resto obtenido de dividir el número mayor entre el más pequeño. (p. 202). Este algoritmo también puede ser implementado en Python.

*Para más información sobre **el Máximo común divisor. Algoritmo de Euclides** consulte el **capítulo 10** en la página **201** del libro **Criptografía sin secretos con Python** de Arboledas, 2017en el siguiente enlace:
https://elibro.net/es/ereader/usanmarcos/106497?page=3

Ataque a la cifra Afín

En este capítulo se pretende que un programa permita a un ordenador encontrar, mediante un ataque de fuerza bruta, las claves de decimación y de desplazamiento con las que se ha cifrado un mensaje.

El espacio de claves en la cifra afín

En la cifra de César, el principal problema es que la clave de cifrado solo se puede utilizar 25 claves en un alfabeto de 26 caracteres. Con la cifra afín aumenta de gran manera el espacio para más claves debido a que ahora hay dos operaciones para encontrar el símbolo de cifrado: un producto y una suma, de modo que el número de claves posibles será el producto de las constantes de decimación y de desplazamiento.
Esto implica que para el alfabeto de 97 símbolos que hemos empleado habrá un número máximo de $K = 97\ 97 = 9409$. claves. Sin embargo, el número real es menor, pues solo son posibles claves de decimación que sean coprimos con la longitud del alfabeto.

Código fuente

En el ordenador, se abrirá el entorno interactivo de Python, haz clic en el menú **File**>**New File**, escribir el código fuente, guardar como y pulsar F5 para ejecutarlo:

Se mostrará algo así:

```
 1. # Fuerza bruta contra la cifra Afín
 2.
 3. import pyperclip, cifraAfin, detectarEspanol, criptomat
 4.
 5. MODO_SILENCIO = False
 6.
 7. def main():
 8.     print('Este programa realiza un ataque por fuerza
bruta contra la cifra Afín')
 9.     criptograma = input('\nIntroduce el criptograma: ')
10.     texto = criptoanalisis(criptograma)
11.
12.     if texto != None:
13.         print('\nCopiando mensaje al portapapeles:')
14.         print(texto)
15.         pyperclip.copy(texto)
16.     else:
17.         print('No fue posible hallar la clave.')
18.
19.
20. def criptoanalisis(mensaje):
21.     print('Buscando...')
```

```
22.     print('(Pulsa Ctrl-C or Ctrl-D para salir.)')
23.
24.     # Comenzamos a recorrer todas las claves
25.     for clave_A in range(2,97):
26.         if criptomat.mcd(clave_A, len(cifraAfin.SIMBOLOS))
!= 1:
27.             continue
28.         for clave_B in range(0,97):
29.             texto = cifraAfin.descifrar_mensaje(mensaje,
clave_A,clave_B)
30.             if not MODO_SILENCIO:
31.                 print('Probando clave (%s, %s)... (%s)' %
(clave_A, clave_B, teXto[:40]))
32.
33.             if detectarEspanol.es_espanol(texto,0.25
[0]:
34.                 # Le permite al usuario comprobar la
solución
35.                 print('\nPosible hallazgo:')
36.                 print('\tClave: (%s, %s)' % (cla ve_A,
clave_B))
```

```
37.                 print('\tTexto plano: ' + tex to[:100])
38.                 print('\nPulsa F para finalizar o Enter
para continuar:')
39.                 response = input('> ')
40.                 if response.strip().upper().star
tswith('F'):
41.                     return texto
42.     return None
43.
44.
45. if __name__ == '__main__':
46.     main()
```

Fuente: Arboledas, 2017

Cuando se ejecute el programa, este pedirá introducir el criptograma. Una vez hecho, pulsar la tecla Enter y el hará el resto. En la pantalla aparecerá algo similar a esto:

```
Este programa realiza un ataque por fuerza bruta contra la
cifra Afín

Introduce el criptograma: ^TXmT$¡\}@X}T$mH\@Xm@$9m$HDT-q-9
ymy$y}$L\}$
TX}$=mX}P-m9$)mlm$X}@-yD$\@$H}TD$-=HDPXm@X}$}@$9m$muX\mu
D@$y-P-%-ym$HDP$}9$1\}p$y}$9m$N\y}@u-m$"mu-D@m9_
Buscando...
```

```
(Pulsa Ctrl-C or Ctrl-D para salir.)
Probando clave (2, 0)... (¿:<v:Q!>~0<~:Qv4>0<v0Q,vQ42:&x&,&|
|Q|-Q)
Probando clave (2, 1)... (nikEi¡OmM_kMi¡Ecm
kE_¡[E¡caiUGU[UKEK¡KM¡)
Probando clave (2, 2)... (>9;u9P =}/;}9Pu3=/;u
P+uP319%w%+%{u{P{}P)
...
Probando clave (4, 3)... (vCD2CO7E6¿D6CO2@E¿D2¿O=2O@?C:3:=:5
5O56O)
Probando clave (4, 4)... (.[\I[gN]MV\M[gIX]V
IVgTIgXW[QJQTQLILgLMg)
Probando clave (4, 5)... (Estas fuentes apuntan la
posibilidad de )

Posible hallazgo:
  Clave: (4, 5)
  Texto plano: Estas fuentes apuntan la posibilidad de que
este material haya tenido un peso importante en la actua

Pulsa F para finalizar o Enter para continuar:
> f

Copiando mensaje al portapapeles:
Estas fuentes apuntan la posibilidad de que este material
haya tenido un peso
importante en la actuacion dirigida por el juez de la Aude
cia Nacional.
```

Fuente: Arboledas, 2017

Cómo funciona el programa *

```
1. # Fuerza bruta contra la cifra Afín
2.
3. import pyperclip, cifraAfin, detectarEspanol, criptomat
4.
5. MODO_SILENCIO = False
```

En la línea 3 se le indicará al programa que importe otros módulos (vistos anteriormente), ya que, sin estos, no funcionará de manera correcta.

Cuando se ejecute el programa, en la pantalla se observará todos los mensajes descifrados con cada una de las claves posibles. Al mostrar estas líneas el programa se ralentiza y para evitar esto se cambia asignación de la variable de la línea 6 `MODO_SILENCIO` a `True`.

```
 7. def main():
 8.     print('Este programa realiza un ataque por fuerza
bruta contra la cifra Afín')
 9.     criptograma = input('\nIntroduce el criptograma: ')
10.     texto = criptoanalisis(criptograma)
11.
12.     if texto != None:
13.         print('\nCopiando mensaje al portapapeles:')
14.         print(texto)
15.         pyperclip.copy(texto)
16.     else:
17.         print('No fue posible hallar la clave.')
```

En la línea 9 se almacena la variable homónima con el criptograma introducido. El código de las líneas 12 y 17 comprueba el contenido de la variable texto. Si no es `None`, la línea 14 muestra el mensaje descifrado y se copia al portapapeles en la 15. En caso contrario, el programa indicará que no ha podido encontrar la clave.

En otras líneas del código fuente se da la **búsqueda exhaustiva de claves** donde se recalca que la función ´principal del programa es la de `criptoanálisis ()` ya que será la responsable de probar una a una todas las claves del algoritmo. Este proceso es algo largo, por lo que, si en algún momento se desea salir, se debe presionar las teclas Ctrl-C (Windows) o Ctrl-D (OS X y Linux).

*Para más información sobre **cómo funciona el programa** consulte el **capítulo 10** en la página **201** del libro **Criptografía sin secretos con Python** de Arboledas, 2017 en el siguiente enlace: https://elibro.net/es/ereader/usanmarcos/106497?page=3

Manejo de excepciones

Una excepción, en un lenguaje de programación, es la indicación de un problema inusual que ocurre durante la ejecución de un programa. El manejo de excepciones permite al usuario crear aplicaciones tolerantes a fallos y que puedan seguir ejecutándose sin verse afectadas por el problema. Python usa objetos para representar condiciones excepcionales. Si no se manejan adecuadamente, el programan terminará abruptamente con un mensaje de error. (Arboledas, 2017, p. 230)

A continuación, se deberá abrir el intérprete de comando y escribir lo siguiente:

```
>>> 3/0
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
```

```
    3/0
ZeroDivisionError: division by zero
```

Como no se puede dividir por 0, el programa lanza un error, sin embargo, cada excepción es una instancia de alguna clase, ente caso (`ZeroDivisionError`). El objetico acá es controlar la excepción y evitar errores.

Python utiliza las instrucciones `try/except`. Para manejar estas excepciones:

```
try:
    3/0
except ZeroDivisionError:
    print('No es posible dividir por 0')
```

Cuando se ejecutan estas líneas, el programa intenta hacer la división indicada, Además se incluyen las funciones `ZeroDivisionError`y`print()`y al realizarse el procedimiento correctamente, el programa terminará de manera abrupta.

La función `es_espanol ()` del módulo detectarEspanol, del que hace uso nuestro programa, evalúa la expresión len (texto) /longitud, donde longitud mide la cantidad de letras que tiene el mensaje, se eliminan signos de puntuación, números y caracteres especiales.

Igualmente, al añadir en el módulo *detectarEspanol* el bloque `try/except`, gestionamos de manera adecuada la posibilidad de que la longitud del mensaje que se le pasa a la función sea cero. (p. 232)

Cifra de sustitución simple

La transposición columnar y la cifra afín poseen miles de claves posibles, pero como hemos visto, son vulnerables frente a un ataque por fuerza bruta. Es por esto que ahora se conocerá un algoritmo inmune a un ataque de estos en menos tiempo y con un coste asumible.

La también llamada alfabeto mezclado, ya que emplea un alfabeto aleatorio formado por la misma cantidad de símbolos y misma longitud que el alfabeto de planos, es un algoritmo sencillo que garantiza la inviabilidad de efectuar un ataque por fuerza bruta. Su tamaño del espacio de claves es tal que, aun teniendo la capacidad de probar un billón de claves por segundo, se necesitaría casi 13 millones de años para

recorrer el espacio de claves completo y, por tanto, tener la certeza de que el criptograma quedaría desvelado. (p. 235)

Para hacer uso de este método, se elige al azar del alfabeto para cifrar la letra A, a continuación, se selecciona aleatoriamente de las otras 25 una nueva letra para la B, y así sucesivamente. El resultado es un alfabeto aleatorio de 26 símbolos.
El número total de claves en esta cifra resulta entonces tan astronómico como 26! = 403 291 461 126 605 635 584 000 000.

Ahora un ejemplo:

Se va a cifrar el mensaje “Atacar a las seis en punto” con la clave SHCLAOKJIFRMDEVYTGUWQPZBNX.

Paso 1: Colocar el alfabeto llano y bajo e’l el de sustitución.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
S	H	C	L	A	O	K	J	I	F	R	M	D	E	V	Y	T	G	U	W	Q	P	Z	B	N	X

Paso 2: Ahora, para cifrar el texto plano, se deberá buscar la letra *a* y se sustituye por su pareja, que es la *s, la t que se cifra con la W* y así sucesivamente, dando como resultado*:* **SWSCSO S MSU UAIU AE YQEWV.**

La ventaja de este método es que no es un desplazamiento como cifra César, sino que es un alfabeto de 26 letras aleatorias. Sin embargo, esto también resulta ser una desventaja ya que memorizar 26 letras sin sentido es bastante complicado.

Cómo funciona el programa

```
1. # Cifra de Sustitución Simple
2.
3. import pyperclip, sys, random
4.
5.
6. LETRAS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

La primera línea es un comentario con lo que hace el programa. La línea 3 importa los tres módulos de los que hará uso el programa.

```
 8. def main():
 9.     print("""\nElije una opcion:
10.   1. Cifrar
11.   2. Descifrar
12.   3. Generar clave""")
13.
14.     opcion = int(input('\nOpcion > '))
```

La función `main()` comienza generando un breve menú con las tres opciones que permite nuestro programa: 1, para cifrar un texto; 2, para descifrar un criptograma y 3, para generar un alfabeto mezclado que pueda servir como contraseña con cualquiera de las dos primeras opciones. En la línea 14 recogemos en la variable opción, como un número entero, la elección del usuario.

```
16.     # Menú de opciones
17.     if opcion == 1:
18.         modo = 'cifrado'
19.         mensaje = input('Mensaje > ')
20.         clave = input ('Clave > ')
21.         comprobarClave(clave)
22.         texto = cifrarMensaje(clave, mensaje)
23.
24.     elif opcion == 2:
25.         modo = 'descifrado'
26.         mensaje = input('Mensaje > ')
27.         clave = input ('Clave > ')
28.         comprobarClave(clave)
29.         texto = descifrarMensaje(clave, mensaje)
30.
31.     elif opcion == 3:
32.         clave = claveAleatoria()
33.         print('Clave: ', clave)
34.         sys.exit()
```

Las líneas 16-34 gestionan las distintas posibilidades contempladas en el menú.
Las líneas 21 y 28 comprueban la validez de la clave utilizada. Es muy fácil equivocarse con un alfabeto mezclado, ya sea porque le falte alguna letra del alfabeto, o porque se repita alguna. Para evitar esto, está la función `comprobarClaVe()` se asegurará de que el alfabeto mezclado sea correcto y en caso contrario abandonará el programa con una advertencia.

```
36.     # Impresión del texto llano o criptograma
37.
38.     print('\nClave %s' % (clave))
39.     print('El mensaje %s es:' % (modo))
40.     print('  ',texto)
41.     pyperclip.copy(texto)
42.     print()
43.     print('El mensaje se ha copiado al portapapeles.')
```

En la línea 38 se muestra el valor de la clave utilizada en el proceso y el mensaje cifrado, o descifrado, se imprime y se copia al portapapeles. Por último, la línea 43 indica que el mensaje se ha copiado al portapapeles.

El método de listas sort ()

```
46. def comprobarClave(clave):
47.     listaClave = list(clave)
48.     listaLetras = list(LETRAS)
49.     listaClave.sort()
50.     listaLetras.sort()
```

En las líneas 47 y 48 la clave y el alfabeto se convierten en una lista por acción de `list()`. El método de listas `sort()` reagrupa los elementos en orden alfabético, por tanto, las listas que contienen las variables `listaLetras` y `listaClave` se ordenan después alfabéticamente invocando el método `sort( )`.

```
51.     if listaClave != listaLetras:
52.         print('Clave incorrecta.')
53.         sys.exit()
```

Una vez ordenadas, los valores de `listaLetras` y `listaclave` deberían ser los mismos. Si es así, entonces tenemos una clave perfectamente válida. Si no son iguales, entonces la línea 51 se evalúa a `True` y el programa con una llamada a `sys.exit ()` indicando que la clave es incorrecta.

Funciones envolventes

```
56. def cifrarMensaje(clave, mensaje):
57.     return mensajeCod(clave, mensaje, 'cifrar'). upper()
58.
59.
60. def descifrarMensaje(clave, mensaje):
61.     return mensajeCod(clave, mensaje, 'descifrar').
lower()
62.
63.
64. def mensajeCod(clave, mensaje, modo):
```

El código fuente empleado para cifrar y descifrar los mensajes en el programa son prácticamente idénticos. Es mejor, por tanto, crear una única función e invocarla dos veces que escribir el código fuente dos veces.

Las funciones envolventes encapsulan el código de otras funciones y devuelven el valor que éstas regresan, a veces, con ligeras modificaciones. En este caso, las funciones `CifrarMensaje()` y `descifrarMensaje()` son las funciones envolventes que encapsulan a la función `mensajeCodi()`, y por tanto, las que devolverán el valor regresado por esta última. En la línea 64, la función `mensajeCod()` es quien verdaderamente cifra o descifra un mensaje en función del valor del parámetro `modo`.

El proceso de cifrado en una cifra por sustitución, es muy simple: para cada letra del mensaje original se halla su posición o índice en el alfabeto original, llamado `LETRAS`, y se reemplaza con el símbolo que ocupa la misma posición en el alfabeto de sustitución, que es clave.

Para descifrar se sigue el proceso opuesto: se encuentra el índice que ocupa el símbolo de la clave y se reemplaza con la letra que ocupa la misma posición en `LETRAS`.

```
73.     # Se recorre uno a uno cada simbolo del mensaje
74.     for simbolo in mensaje:
75.         if simbolo.upper() in carsA:
76.             # cifra/descifra el simbolo
77.             Indice = carsA.find(simbolo.upper())
```

En línea 74 hay un bucle for que recorrerá uno a uno los símbolos del mensaje. Como `LETRAS` y clave se introducen en mayúsculas, la línea 75 comprueba si la forma mayúscula de cada letra del mensaje se encuentra en `carsA`. Si es así, se obtiene la posición que ocupa y se almacena en la variable Índice en la línea 77.

Los métodos de cadena isupper() e islower()

Estos métodos permiten que se logre conocer si los caracteres que forman una cadena se encuentran todos en mayúscula o minúsculas. Por una parte el método `isupper()` devolverá `True` cuando no haya ningún carácter en minúscula y por otra parte, `islower()` devolverá `True` si no contiene ninguna mayúscula. (p. 244)

Ejemplos vistos desde el terminal interactivo de Python:

```
>>> 'Hola mundo'.isupper()
False
>>> 'Hola mundo'.islower()
False
>>> 'HOLA MUNDO'.isupper()
```

```
True
>>> 'hola mundo'.islower()
True
>>> 'DS 3'.isupper()
True
>>> '45@'.islower()
False
>>>
```

En el código fuente se vería así:

```
78.             if simbolo.isupper():
79.                 texto += carsB[Indice].upper()
80.             else:
81.                 texto += carsB[Indice].lower()
```

Si `simbolo` es una letra mayúscula, entonces concatenamos el carácter en `carsB [Indice]` a texto.
Si `simbolo` es un número o un signo de puntuación, entonces la condición de la línea 78 sería `False` y se ejecutaría la línea 81.

Asimismo, se puede generar una **clave pseudoaleatoria** cuando una se debe teclear en una cadena aleatoria que contenga una o solo una letra en un conjunto de 97 símbolos. Esta clave se crea con la función `random.shuffle()`, misma que reorganiza los elemento de forma pseudoaleatoria.

Cómo cifrar otros símbolos

En páginas anteriores se ha visto cómo por medio de la sustitución simple se logran cifrar solo letras del alfabeto, pero como se cifran otros elementos que no sean letras.

Para ello, haz el siguiente cambio en la línea 6 del código fuente del programa:

```
6. LETRAS = r"""!¡"#$%&'()*+,-./0123456789:;<=>¿?@ABCDEFGHIJK
LMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~"""
```

Se deberá usar las triples comillas para evitar el uso de caracteres de control, lo cual facilita la escritura. Además, el alfabeto de sustitución y ya no tienen sentido las distinciones efectuadas entre formas mayúsculas y minúsculas. Ahora, cuando se ejecute el programa *sustitucionpy* cualquier criptograma tendrá una apariencia completamente embrollada:

```
Clave v3UQBpcXLEs=rH¿-"'hVC>¡x64YTu@(]8d&5F\[1IRm2$
z9a:#qykeW;)0<nP!
Klb%gMAi|+^`7O/~?}_NoZwj,GJ{*fDS
El mensaje cifrado es:
     &O  /A~?o  /AM^b  M?gA~b  MA  %b~g?o  ^~ZAN~bg^?~bOAo
AoZá~ bOANZb~M? b owo gO^A~ZAo MAO MAZAN^?N? _wA }wAMA
owiN^N Ob Ag?~?/íb Ao}bñ?Ob ZNbo Obo AOAgg^?~Ao |A~ANbOAo

El mensaje se ha copiado al portapapeles.
>>>
```

Ataque a la cifra de sustitución simple

Para romper todos los métodos de cifrado anteriormente mencionados, se ha utilizado el ataque por fuerza bruta, sin embargo, para la cifra de sustitución simple no es factible debido al enorme espacio de claves que lo caracteriza. Es por esto que, el ataque que se utilizará para romper esta cifra será el **ataque de diccionario.**

Según Arboledas, 2017, el ataque de diccionario es una técnica que consiste en

> intentar hallar una contraseña probando todas las palabras de un diccionario. A diferencia de un ataque por fuerza bruta, donde se busca sistemáticamente en la totalidad del espacio de claves, un ataque por diccionario solo prueba aquellas posibilidades que son más factibles de ocurrir. Esto se debe a la tendencia de la gente a usar palabras de su propia lengua o pequeñas variantes como contraseñas para que la clave sea fácil de recordar. (p. 251)

A continuación, se observará cómo implementar esta técnica en Python con la cifra de sustitución simple, tomando en cuenta que la clave debe ser una palabra que se encuentre dentro del diccionario de la lengua española.

Implementación del ataque *

El ataque de diccionario consiste en probar todas y cada una de las palabras presentes en el mismo hasta dar con una que rompa el criptograma. En el peor de los casos tendremos que probar 82.000 palabras, las quecontiene `diccionari0.lxl`. por lo que si la contraseña es una palabra que no se encuentra en el diccionario, la ofensiva fallará.

Para la implemntación de este ataque se necesitan los módulos: *pyperclip. py, sustitucionSimple.py* y *detectarEspanol.py*,

El **código fuente** consta de 62 líneas, mismas que son necesarias para que el ataque de diciconario se realice con éxito. Dentrp de esta implementación, se abordan el ***método de listas remove ()*** (línea 29) y ***la funciónporcentaje_palabras*** (línea 25), pasos importantes de conocer por el usuario que ejecute el programa en un ordenador.

*Para más información sobre **la implementación del ataque a la cifra de sustitución simple** consulte el **capítulo 13** en la página **252**el libro **Criptografía sin secretos con Python** de Arboledas, 2017 en el siguiente enlace:
https://elibro.net/es/ereader/usanmarcos/106497?page=3

Todas las cifras que ya se han visto, tienen la característica de ser monoalfabéticas. Estas gobernaron por casi mil años el mundo de la escritura a pesar de saber que no eran muy seguras y que con ayuda de libros de código, se podía cifrar y decifrar los mensajes que contenían con facilidad.

Ya para el siglo XVI el espionaje en Europa avanzaba con gran rapidez por lo que se comienza a poner más en práctica las cifras polialfabéticas. (Arboledas, 2017 citando a Kahn, 1996, p. 263).

En las siguinetes páginas se conocerá un poco de algunas de estas cifras polialfabéticas:

La cifra de Battista Bellaso

La cifra es una sustitución polialfabética y a diferencia del sistema de discos de Alberti, es todavía periódica, pero el uso de contraseñas largas la hacían más segura, incluso aunque la tabla recíproca en la que se basaba fuera de

> dominio público, lo que esta en concordancia con el principio de Kerckhoffs. (Arboledas, 2017, p. 264)

El mecanismo para cifrar un texto llano mediante este método, conciste en elaborar un tabla recíproca de cinco alfabetos mezclados, donde la tabla comienza en la primera palabra:

- Si el número de letras es par, se distribuye mitad a mitad entre las dos líneas del alfabeto.
- Si es impar, se sitúa en la línea de arriba hasta la letra que se encuentra en la mitad de la palabra y el resto en la de abajo.

I D K N T Z	i o a b c d f g h j k l m
	v e n p q r s t u w x y z
O F L P U	i o a b c d f g h j k l m
	z v e n p q r s t u w x y
A G M Q W	i o a b c d f g h j k l m
	y z v e n p q r s t u w x
B H V R X	i o a b c d f g h j k l m
	x y z v e n p q r s t u w
C J E S Y	i o a b c d f g h j k l m
	w x y z v e n p q r s t u

Fuente: Arboledas, 2017 p. 264

La cifra Vigenere

Tras Bellaso apareció Blaise de Vigenere quien conoció de primera mano los escritos del mismo Bellaso. A pesar de que al principio el interés de Vigenere fue meramente práctico, después se metió de lleno al estudio de la cifras. Actualmente existe una cifra de Vigenere que no tiene ninguna similitud con la que se creó originalmente en 1586, pero de igual menra se debe estudiar de manera brve en este documento. (p. 283)

La primera cifra de Vigenere

Esta cifra posee muchas similitudes a la cifra de Bellaso y la segunda mitad de cada alfabeto se desplaza una unidad hacia la derecha, como en la cifra Bellaso, sin embargo, la cifra de Vigenere emplea 10 alfabetos generados sin contraseña.

El proceso de cifrado o descifrado se realiza mediante una palabra clave. Con esta contraseña sitúa su primera letra sobre la primera letra del mensaje, su segunda letra sobre la segunda letra y así sucesivamente.

A continuación, se cifra letra a letra según la tabla recíproca de Vigenere con el mensaje *Au nom de l'élernel soit mon commencement* con la contraseña **LA NUIT CLAIRE.**

A	a	b	c	d	e	f	g	h	i	l
B	m	n	o	p	q	r	s	t	u	x
C	a	b	c	d	e	f	g	h	i	l
D	x	m	n	o	p	q	r	s	t	u
E	a	b	c	d	e	f	g	h	i	l
F	u	x	m	n	o	p	q	r	s	t
G	a	b	c	d	e	f	g	h	i	l
H	t	u	x	m	n	o	p	q	r	s
I	a	b	c	d	e	f	g	h	i	l
L	s	t	u	x	m	n	o	p	q	r
M	a	b	c	d	e	f	g	h	i	l
N	r	s	t	u	x	m	n	o	p	q
O	a	b	c	d	e	f	g	h	i	l
P	q	r	s	t	u	x	m	n	o	p
Q	a	b	c	d	e	f	g	h	i	l
R	p	q	r	s	t	u	x	m	n	o
S	a	b	c	d	e	f	g	h	i	l
T	o	p	q	r	s	t	u	x	m	n
V	a	b	c	d	e	f	g	h	i	l
X	n	o	p	q	r	s	t	u	x	m

Fuente: Arboledas, 2017

Se procedería así:

LA	NUI	TC	L AIRELAN	UITC	LAI	RELANUITCLAI
Au	nom	de	l'éternel	soit	mon	commencement

Ahora se deberá buscar en la primera columna de la izquierda la letra de la clave y en el alfabeto la letra del texto claro. El caracter cifrado es aquél con el que se empareja en el alfabeto la letra del texto llano. Por lo que, para la letra L en la clave y la *a* en eltexto llano, resulta *s*. Para la letraA en la clave y la *u* en el texto plano, tenemos la I, y asi sucesivamente.

Vigenere empleaba originalmente **cuatro letras** arbitrarias al principio del criptograma,
Por ejemplo *fsbm*; así como las letras *Z e y* como nulas, separando las palabras del texto. De este modo, el criptograma obtenido del mensaje anterior era escrito por

Vigenere de la siguiente manera: *fsbmsiygbezrpzrqblhquzfgmizecfyreeaxausbmbb.*Cómo se logró observar, se trata de una cifra muchísimo más segura que cualquier otra monoalfabética,

La cifra de autoclave

La cifra de autoclave, se utiliza cuando resulta difícil generar una clave distinta con cada interlocutor al momento de cifrar el mensaje, lo que no ocurre con la cifra anterior, ya que posee una seguridad elevanda, siempre y cuando se utilice una clave compleja.

> Arboledas, 2010 indica que la cifra de autoclave emplea métodos trabajan que con una clave inicial a la que se añade el texto claro o el criptograma. De este modo, se emplea una contraseña sencilla y larga como para que la labor de descifrado sea enormemente compleja si no se conoce la clave o semilla inicial. Sin embargo, no es para el criptoanalista una labor imposible, ni muchísimo menos. Esto es así porque el texto llano esta formado por palabras del lenguaje natural y, por tanto, predecibles. (p. 285)

Vigenere explicaba el método de autoclave de la siguiente manera:

Imagina que deseas transmitir el mensaje *Las últimas noticias confirman un atentado* empleando como contraseña LA LUNA NUEVA. Lo primero es anexar a la clave el texto llano, que quedaría así: LALUNANUEVALASULTIMAS NOTICIASCONFIRMANUNATENTADO. Ahora se cifraría del modo habitual con el metodo descrito anteriormente.

LAL	UNANUEV	ALASULTI	MASNOTICI	AS	CONFIRMA
Las	ultimas	noticias	confirman	un	atentado

Utilizandoel programa *Vigenere1586.py*se logra comprobar que el criptograma obtenido sería este:

rmathhplufi/bghmpqoaytclmodexfzilzxdxdbpuc

El proceso opuesto es algo mas complejo, pues el destinatario no conoce el texto plano y, por tanto, la totalidad de la clave. Sin embargo, sí sabe cual es su comienzo:

LA LUNA NUEVA. Así que su primer paso ha de ser descifrar el criptograma hasta donde conoce:

LAL	UNANUEV	A			
rma	hqhpluf	bghmpqoa	tclmodexf	il	xdxdbpuc

Si se descifra estas primeras once letras obtiene: Las últimas n, que se añadirían a la clave:

LAL	UNANUEV	**ALASULTI**	**MASN**		
rma	hqhpluf	bghmpqoa	tclmodexf	il	xdxdbpuc

Y así seguiría descifrando y añadiendo letras a la clave hasta conocer el texto.

La cifra indescifrable

Esta última es de las que más seguridad aporta a un proceso comunicativo, con tal solo decir que siempre se la ha conocido como *lechiffreíndéchiffrable* y que por casi 300 años fue inmune a cualquier ataque. Sin embargo, los estudios independientes de Charles Babbagey Friedrich Kasiskí, consiguieron generalizar una ofensiva exitosa contra la cifra Vigenere, incluyendo el uso de autoclaves. (p. 288)

*Para más información sobre **la cifra Bellaso, Vigenere, autoclave y descifrado** consulte los **capítulos14 y 15** en las páginas **263 a la 288** del libro **Criptografía sin secretos con Python** de Arboledas, 2017 en el siguiente enlace: https://elibro.net/es/ereader/usanmarcos/106497?page=3

Referencias bibliográficas

Arboledas, D. (2017). Criptografíasin secretos con Python. España: RA-MA Ediciones. Recuperado de: https://elibro.net/es/ereader/usanmarcos/106497?page=3

San Marcos

www.usanmarcos.ac.cr

San José, Costa Rica