CIFRADO DE HILL

Rubén Martín Hidalgo - Rafael Lachica Garrido

---

1. Cifrado de Hill

Fue Inventado por Lester S. Hill en 1929, y fue el primer sistema criptográfico polialfabético que era práctico para trabajar con mas de tres símbolos simultáneamente. Este sistema es polialfabético, pues puede darse que un mismo carácter en un mensaje a enviar se encripte en dos caracteres distintos en el mensaje encriptado.

Tenemos la siguiente correlación de letras y números:

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Se elige un entero d que determina bloques de d elementos que son tratados como un vector de d dimensiones. Se elije de forma aleatoria una matriz de d × d elementos los cuales serán la clave a utilizar. Los elementos de la matriz de d × d serán enteros entre 0 y 25, además la matriz M debe ser invertible en $\mathbb{Z}_{26}^{n}$.

Para la encriptación, el texto es dividido en bloques de d elementos los cuales se multiplican por la matriz d × d. Todas las operaciones aritméticas se realizan en la forma módulo 26, es decir que 26=0, 27=1, 28=2 etc.

Dado un mensaje a encriptar debemos tomar bloques del mensaje de "d" caracteres y aplicar:
$M \times P_i = C$, donde C es el código cifrado para el mensaje P_i

2. Código Fuente Cifrado de Hill

Este código lo que hace es cifrar un mensaje con el cifrado de Hill. Para ello usa las matrices para encriptar según el algoritmo de Hill. Explicaremos poco a poco las principales funciones del código

```
import string
class Hill:
    #Constructor
```

```
    #parámetros: la clave (passkey), desplazamiento a la izquierda y caracter a
reemplazar
    def __init__(self, passkey, leftmul = False, replacespacechar = ' '):
        self.passkey = [ord(c) - ord('A') for c in passkey[:4].upper()]
        #pasa a mayúsculas todo el texto

        self.leftmul = leftmul
        self.replacespacechar = replacespacechar
        #actúa de constructor, copia los parámetros dentro de las variables de
clase

        self.dekey = [self.passkey[3], -self.passkey[1], -self.passkey[2],
                      self.passkey[0]]
        det = inv_mod(self.passkey[0] * self.passkey[3]
                      - self.passkey[2] * self.passkey[1], 26) % 26
        self.dekey = [n * det % 26 for n in self.dekey]
        #utiliza la clave para cifrar aplicando la inversa de la matriz de la
clave,     el inv_mod de la clave (passkey)
        #divide la clave en 3 bloques de 26 letras de hay el módulo %26

#Función preprocesamiento comprueba que no haya espacios y que esté todo en
Mayúsculas
    def _preprocess(self, s):
        return "".join(c if c in string.ascii_uppercase
                       else self.replacespacechar if str.isspace(c)
                       else "" for c in s.upper())
#Si hay algún espacio lo reemplazo por el carácter vacío sin espacios

#Función para encriptar que toma como parámetros 2 carácteres, ya que tomamos
bloques de código de 2x2, y aquí ciframos carácter por carácter uno por matriz.
Para ello correlacionamos el carácter 1 y 2 con las letras del alfabeto, ejemplo A
= 0
    def encrypt_chars(self, ch1, ch2):
        c1 = ord(ch1) - ord('A')
        c2 = ord(ch2) - ord('A')
        if self.leftmul:
            return (chr(ord('A')
                    + (c1 * self.passkey[0] + c2 * self.passkey[1]) % 26),
                    chr(ord('A')
                    + (c1 * self.passkey[2] + c2 * self.passkey[3]) % 26))
#Aplicamos el algoritmo de Hill de cifrado con bloques de matrices definidos antes
y le aplicamos el módulo 26 de nuestro alfabeto. En este caso si hay
#desplazamiento a la izquierda.
        else:
            return (chr(ord('A')
                    + (c1 * self.passkey[0] + c2 * self.passkey[2]) % 26),
                    chr(ord('A')
                    + (c1 * self.passkey[1] + c2 * self.passkey[3]) % 26))
#Si hay desplazamiento aplicamos su correspondiente algoritmo, teniendo como
referencia la primera letra del alfabeto A

#Función que llama al cifrado de carácter por carácter, para una cadena source
    def encrypt(self, source):
        s = self._preprocess(source)
```

```
        l = len(source)
        return "".join("".join(self.encrypt_chars(source[i], source[i+1]))
                       for i in range(0, l-1, 2))
```

#Función que desencripta el mensaje, carácter por carácter, opera casi igual que el cifrado, asignando un número a cada carácter y aplicando el algoritmo de cifrado de Hill con la clave aplicada para descifrar, la dekey

```
    def decrypt_chars(self, ch1, ch2):
        c1 = ord(ch1) - ord('A')
        c2 = ord(ch2) - ord('A')
        if self.leftmul:
            return (chr(ord('A')
                    + (c1 *  self.dekey[0] + c2 *  self.dekey[1]) % 26),
                    chr(ord('A')
                    + (c1 *  self.dekey[2] + c2 *  self.dekey[3]) % 26))
        else:
            return (chr(ord('A')
                    + (c1 *  self.dekey[0] + c2 *  self.dekey[2]) % 26),
                    chr(ord('A')
                    + (c1 *  self.dekey[1] + c2 *  self.dekey[3]) % 26))
```

#Llama a la función que descifra con la clave letra por letra, para la cadena source

```
    def decrypt(self, source):
        s = "".join(c.upper() for c in source if c in string.ascii_letters)
        l = len(source)
        return "".join("".join(self.decrypt_chars(source[i], source[i+1]))
                       for i in range(0, l-1, 2))
```

#Inverso de un número n, con módulo p, usado para generar la matriz inversa

```
def inv_mod(n, p):
    x = 0
    lastx = 1
    y = 1
    lasty = 0
    while p != 0:
        quotient = n // p
        (n, p) = (p, n % p)
        (x, lastx) = (lastx - quotient*x, x)
        (y, lasty) = (lasty - quotient*y, y)
    return lastx

if __name__ == '__main__':
    en = [('BLAZE OF GLORY', 'jbvi', True),
          ('COMPLETE AND PROPER PACKAGE', 'NTCR', True),
          ('ESOTERIC TOPIC OF RESEARCH', 'BYGP', False)]

    for pt in en:
             cadena = Hill(pt[1], pt[2], '').encrypt(pt[0])
             print cadena
             print Hill(pt[1], pt[2], '').decrypt(cadena)
```

Este código nos genera el siguiente mensaje cifrado:
UFZSYOZXJFVD
GIZTLMLCNNMBTMLUMDMIAUYC
ICYXCNUOZQLMIYDLICESDW

Vemos aquí una muestra de su ejecución :

3. Ataque fuerza bruta cifrado de Hill

```
import string, sys

#Primero pasamos a mayúsculas todo el texto y eliminamos los espacios
#Después dividimos la mitad del mensaje en una matriz
A_0, A_1 = 0, 1
ic = lambda text: sum(((text.count(l) * (text.count(l) - 1)) / float(len(text) *
(len(text) - 1))) for l in string.uppercase)
normalize = lambda text: filter(lambda l: l in string.uppercase, text.upper())
ismatrice = lambda m: (m[0] * m[3] - m[1] * m[2]) % 2 and (m[0] * m[3] - m[1] *
m[2]) % 13

#Función que devuelve la matriz inversa de los 25 primeros números primos
def matriceinv(m):
	a, b, c, d = m
	k = (a * d - b * c)
	for x in (1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25):
		if k * x % 26 == 1:
			break
	return (d * x % 26, -b * x % 26, -c * x % 26, a * x % 26)

#Función desencriptar, que dado un texto obtiene los números primos para descifrar
el texto,obtiene los números primos, de una letra ya que le damos un número a cada
letra del 0 al 26
def decrypt(text, m, z):
	a, b, c, d = m
	letters = [ord(l) - 65 + z for l in text]
	decrypted = []
	i = 0
	while i < len(letters):
		j, k = letters[i], letters[i + 1]
		p = (a * j + b * k) % 26
		q = (c * j + d * k) % 26
```

```
            decrypted += [p, q]
            i += 2
      return ''.join(chr(65 + l - z) for l in decrypted)
```

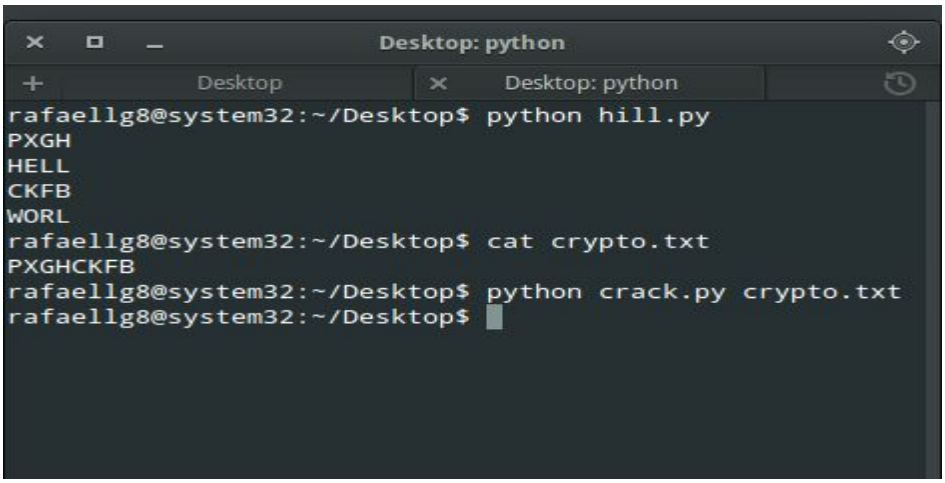
#Método para descifrar por fuerza bruta cada matriz, vamos barriendo las matrices y vamos aplicando la matriz inversa para cada matriz m con los elementos que vamos obteniendo barriendo la matriz, de ahí m = [a,b,c,]

```
def bruteforce(text, A = 0):
      count = 0
      ICMIN = 0.07
      for a in range(26):
            for b in range(26):
                  for c in range(26):
                        for d in range(26):
                        m = [a, b, c, d]
                        if ismatrice(m):
                              m = matriceinv(m)
                              dec = decrypt(text, m, A)
                              if ic(dec) >= ICMIN:
                                    print "[+] Possible text found -- Matrice
inverse: %s -- IC = %s\n%s\n" % (repr(m), ic(dec), dec)
                                    count += 1
      print "\n[+] End of attack - %s possible plaintexts found" % count

def main():
      if len(sys.argv) != 2:
            print "[-] Usage: breakhill.py crypto.txt [> log.txt]"
            sys.exit()

      try:
            text = normalize(open(sys.argv[1], 'rb').read())

      except:
            print "[-] Error while trying to open file"
            sys.exit()

      bruteforce(text, A_1) # A_0 if A=0

if __name__ == "__main__":
      main()
```

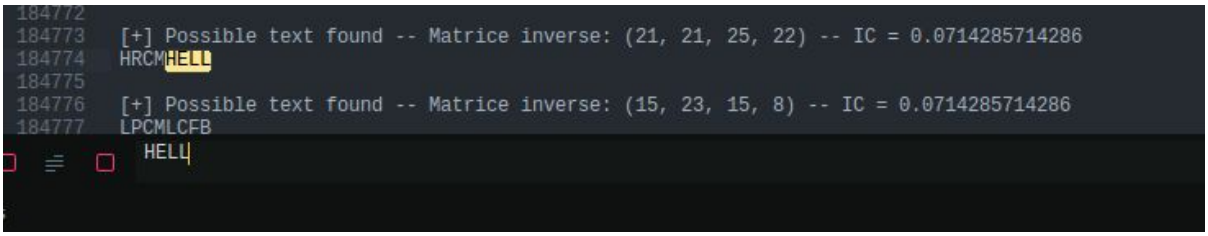
Este código va probando sobre matrices de 2x2, las 26 letras del alfabeto sobre el código encriptado. Esto lo podemos ver en el segmento donde está la función bruteforce, que prueba una matriz de 2x2 con 26 posibilidades cada una.

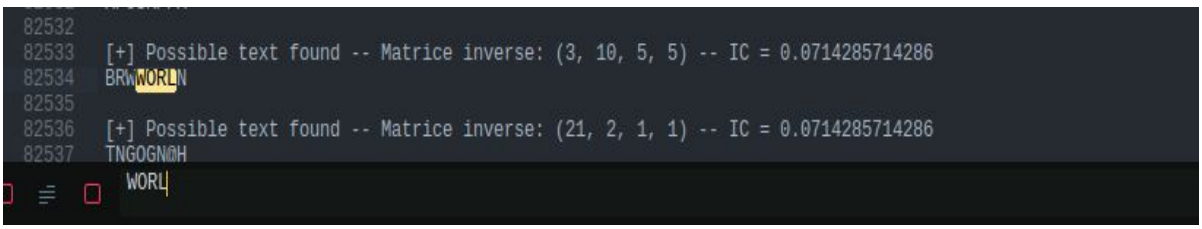
4. Ataque fuerza bruta cifrado de Hill - Pruebas

Generamos el cifrado, guardamos el resultado en crypto.txt y crackeamos:



Obtenemos los fragmentos HELL y WORL, justo los que nos ha cifrado ya que el cifrado lo recorta porque no entra en la matriz de forma adecuada.





En otro ataque, no hemos conseguido descifrar el siguiente ejemplo:

```
romi@ubuntu:~/SPSI$ python hill.py
WNMAMAMSCUGOGC
THISISAMESSAGE
GIZTLMLCNNMBTMLUMDMIAUYC
COMPLETEANDPROPERPACKAGE
ICYXCNUOZQLMIYDLICESDW
ESOTERICTOPICOFRESEARC
UQYVQWFUKQIXQNFUDCZNYMFXCN
WEARECHECKINGTHEHILLCIPHER
romi@ubuntu:~/SPSI$
```

```
if __name__ == '__main__':
    en = [('THIS IS A MESSAGE', 'jbvi', True),
          ('COMPLETE AND PROPER PACKAGE', 'NTCR', True),
          ('ESOTERIC TOPIC OF RESEARCH', 'BYGP', False),
          ('WE ARE CHECKING THE HILL CIPHER', 'BYGP', False)]

    for pt in en:
        cadena = Hill(pt[1], pt[2], '').encrypt(pt[0])
        print cadena
        print Hill(pt[1], pt[2], '').decrypt(cadena)
```

En el fichero crypto.txt metemos “UQYVQWFUKQIXQNFUDCZNYMFXCN” que es el texto cifrado de “WE ARE CHECKING THE HILL CIPHER”.

```
romi@ubuntu:~/SPSI$ cat crypto.txt
UQYVQWFUKQIXQNFUDCZNYMFXCN
romi@ubuntu:~/SPSI$ python breakhill.py crypto.txt > mensaje.txt
[+] Possible text found -- Matrice inverse: (11, 19, 14, 7) -- IC = 0.0707692307692
HWQJ@IWWBMIHKXWWWYFTBYBRMJ

[+] Possible text found -- Matrice inverse: (1, 7, 24, 19) -- IC = 0.0738461538462
JUWDVMWW@OUVKXWWYWTFLORBW@

[+] Possible text found -- Matrice inverse: (24, 19, 1, 7) -- IC = 0.0738461538462
UJDWMVWWO@VUXKWWWYFTOLBR@W

[+] Possible text found -- Matrice inverse: (14, 7, 11, 19) -- IC = 0.0707692307692
WHJQI@WWMBHIXKWWYWTFYBRBJM


[+] End of attack - 1584 possible plaintexts found
```

Tras ejecutar el programa de ataque, nos crea 1584 textos planos posibles, pero ninguno con el mensaje que buscamos.

Después hemos intentado realizar el ataque desde Cryptool, sin conseguir descifrarlo tampoco. Para ello he metido como texto claro y cifrado conocidos los siguientes:

Texto claro: ESOTERIC TOPIC OF RESEARCH
Texto cifrado: ICYXCNUOZQLMIYDLICESDW

Mientras que el texto cifrado a descifrar sigue siendo el mismo de antes:
UQYVQWFUKQIXQNFUDCZNYMFXCN

Tras realizar el análisis, nos muestra la matriz de clave que vemos en la captura anterior.
Copio la clave y la pego al descifrar el texto cifrado, obteniendo esto:
ISGNKYJOAIOJKJJOXIZFGYJTEV

Referencias:

https://en.wikipedia.org/wiki/Hill_cipher

http://cajuna.jimdo.com/cifrado-de-hill/

http://www.deic.uab.es/material/26118-09CifraClasica.pdf

https://code.google.com/p/criptografia/source/browse/trunk/criptografia/Criptoanalisis/src/Ciphers/Hill.java?r=3