Fundamentos de Criptografía: Práctica 1

David Quintana
Alfonso Carvajal

Introducción

En esta primera práctica vamos a programar los métodos clásicos de cifrados, éstos son cifrados por sustitución (monoalfabético y polialfabético) y cifrados por transposición. Además, trataremos el criptoanálisis de los mismos.
Para ello, lo primero que hemos implementado es el algoritmo de Euclides, el cual se encarga de encontrar el máximo común divisor de dos números enteros. La función *Euclidean()* es la que realiza dicha tarea, apelando a la recursividad. En ella, comprobamos si a=0, en cuyo caso devolvemos b como mcd. En caso contrario, calculamos el módulo (b%a) y llamamos a la función recursivamente ahora con módulo y a (esto se realiza hasta que se cumpla la condición del if anteriormente comentada).

```
void euclidean(mpz_t resultado, mpz_t a, mpz_t b){
  mpz_t modulo;
  mpz_init(modulo);

  if(mpz_sgn(a)==0){
    mpz_set(resultado,b);
    mpz_clear(modulo);
    return;
  }
  mpz_mod(modulo,b,a);
  euclidean(resultado,modulo,a);
  mpz_clear(modulo);
}
```

Además, hemos implementado la función *euclideanExtended()*, que además de calcular el mcd de dos números enteros, calcula también los coeficientes de la identidad de Bezout, esto es x e y de la fórmula: "ax + by=gcd(a,b)". En ambas funciones, se ha empleado la librería GMP, la cual nos ha complicado ligeramente la tarea, puesto que era la primera vez que la usábamos.

Sustitución Monoalfabético

Criptografía: Método Afín

Una vez hecho esto, comenzamos con el desarrollo del método afín, para el cual lo primero es comprobar que el máximo común divisor de a y el módulo m es 1, puesto que en caso contrario no existiría inverso multiplicativo y no podríamos descifrar el texto. El bucle for cifra cada caracter del texto plano con la fórmula *caracter_cifrado = (a*caracter_plano[i] + b) % m*, los cuales son leídos/escritos en los ficheros pasados como argumento. La librería GMP complica un poco más el proceso puesto que tenemos que desde la terminal recibimos número enteros y tenemos que iniciar un mpz por cada valor que recibimos (módulo, a y b). Hay que tener en cuenta que nuestro alfabeto comienza en el caracter ascii 65, de ahí la línea *mpz_add_ui(total,total,65)*, que se asegura que los diferentes módulos resultantes

comiencen a partir de tal valor (hasta un máximo de 26, que es la cantidad de caracteres en el alfabeto castellano, sin tener en cuenta la Ñ).

```
for (i=0; i<strlen(textoPlano); i++){
  x = textoPlano[i];
  mpz_mul_ui(a_aux,a1,x);
  mpz_add(a_aux,a_aux,b1);
  mpz_mod(total,a_aux,m1);
  mpz_add_ui(total,total,65);
  textoCifrado[i] = mpz_get_ui(total);
  fprintf(salida,"%c", textoCifrado[i]);
}
```

Una vez cifrado nuestro texto, hacemos la acción complementaria. Leemos el fichero cifrado y lo guardamos en una cadena. Calculamos el inverso multiplicativo de a (gracias a mpz_invert contenida en la librería GMP), el cual sabemos que existe porque ya lo hemos comprobado en la función anterior. Realizamos ahora la resta del 65 y procedemos a descifrar con la fórmula *caracter_descifrado = (a-1*(caracter_cifrado[i] - b) % m*. Una peculiaridad que tenemos que tener en cuenta es que si obtenemos un valor inferior a 65, sumamos tantas veces el módulo como sea necesario para sobrepasar dicho valor (la imagen ilustra perfectamente lo que queremos decir). Una vez finalizado, obtendremos los ficheros con las salidas correspondientes y podremos leer nuestro texto descifrado.

```
for (i=0; i<strlen(textoDescifrado); i++){
  x = textoDescifrado[i]-65;
  mpz_ui_sub(b_aux,x,b1);
  negativo = mpz_get_ui(b_aux);
  while(negativo<0){
    mpz_add(b_aux,b_aux,m1);
    negativo += modulo;
  }
  mpz_mul(mul,inverso,b_aux);
  mpz_mod(total,mul,m1);
  sumando = mpz_get_ui(total);
  while(sumando<65){
    sumando += m;
  }
  textoDescifrado[i] = sumando;
  fprintf(salida,"%c", textoDescifrado[i]);
}
```

Criptoanálisis: Método Afín no Trivial

Sabemos que la robustez del cifrado afín reside en la cantidad de claves que podemos generar, y esta viene dada por la fórmula cardinal(K) = m * funciónEuler(m). En el ejercicio a, teníamos 27 * 18, por tanto es bastante intuitivo pensar que si aumentamos la m, aumentaremos también su función de Euler dándonos un número más grande y por tanto más seguridad ya que poseemos más claves.
Por tanto, la decisión para este apartado b ha sido aumentar el alfabeto (de caracter ASCII 65 al 32), para tener más posibilidad de caracteres e introducir nuevos símbolos. Así, sugerimos por pantalla al usuario que elija un modulo considerablemente elevado, para que la función de euler sea grande, obteniendo así un número elevado de claves.

Sustitución Polialfabético

Método de Vigenere

El método de vigenere es un método de sustitución que parte del tradicional de César pero con la diferencia de que una letra puede ser sustituida por muchas letras (no solo por una). En el caso de Vigenere, se elige una palabra clave y se concatena muchas veces hasta cubrir todo el texto. A partir de esto, simplemente se suman los símbolos del mensaje Módulo el número de caracteres del alfabeto que se está utilizando.
Imaginando que nuestro mensaje fuera:

NOS VEMOS EN LA ESQUINA A LAS CINCO DE LA TARDE EL DOMINGO.

Podríamos elegir una clave como:

TRUCO

Y repitiéndola hasta cubrir el mensaje:

NOS VEMOS EN LA ESQUINA A LAS CINCO DE LA TARDE EL DOMINGO.
TRU COTRU CO TR UCOTRUC O TRU COTRU CO TR UCOTR UC OTRUCOT

Lo cual convertiría nuestro texto en:

GFM XSFFM GB ER YUENZHC O ERM EWGTI FS ER NCFWV YN RHDCPUH.

En nuestro caso, solo trabajamos con mayúsculas. Para eso tenemos una función llamada “Capitalize” que coge un texto, y lo transforma en mayúsculas, dejando igual todos los caracteres que no sean letras.

```
char *capitalize(FILE *F_IN, char *fname){
  char *cap_fname, c;
  FILE *F_CAP;
  cap_fname = (char *)malloc(sizeof(char)*strlen(fname)+5);
  strcpy(cap_fname, "cap_");
  strcat(cap_fname, fname);
  F_CAP = fopen(cap_fname, "w");

  while(EOF != (c = fgetc(F_IN)))
    fputc(toupper(c),F_CAP);

  fclose(F_CAP);
  return cap_fname;
}
```

Para ejecutar nuestro programa metemos en terminal:

./v -C -k TRICKS -i 1984.txt -o c_1984.txt

En este caso la palabra clave es “TRICKS” y el texto a cifrar es 1984.txt.

```
We will be using the novel 1984 for these experiments
Showing the first lines of the book
Title:      Nineteen eighty-four
Author:     George Orwell (pseudonym of Eric Blair) (1903-1950)
PART ONE
Chapter 1

It was a bright cold day in April, and the clocks were striking thirteen.
Winston Smith, his chin nuzzled into his breast in an effort to escape the
vile wind, slipped quickly through the glass doors of Victory Mansions,
though not quickly enough to prevent a swirl of gritty dust from entering
along with him.
```

```
MZBNO:      FBEMVOWG VQIRLR-WWWB
SNKPQB:     YXFZIO GKNMNV (HLVCFYFRD WH OJBT JNKAK) (1903-1950)
GITD GGV
KJKHMVZ 1

KD OTJ I DBAZYB EYDW UIA SF TGZKV, SGU BJO UEFKMC OXIM UDJBBQPQ LAZZVOWG.
NQPCLHE AOSLA, YQU MZBE VWJREVL KXLH YQU LJXRAV SF TE MHPGKK BQ OKVRXG DZX
MQNO OBEL, UVAIGMF AMBTSNI LAIWWQZ MYM IVSLJ LQYJL FN XSUMFZA WSGJQQXK,
MYWWQZ GFB SEAVBTA OFHLOJ DG IIMXOFM R AYSJE FN IBAMKG FEKM WZQW WGKMTSFZ
RTQXY PZBJ RAF.
```

La clave de todo el algoritmo es realmente sencilla:

```
    while(EOF != (c = fgetc(F_IN)))
      if(c >= 'A' && c <= 'Z')
        fprintf(F_OUT, "%c", (k[i++%key_len] + c - 2*'A') % total_letters + 'A');
      else
        fprintf(F_OUT, "%c", c);
      }
```

Esto es la suma modular de las letras del texto con las letras de la palabra clave (ignorando los símbolos que no sean letras para mantener la estructura visual del documento).
Aquí tenemos varios ejemplos:

```
EXP 2:
 Encrypting with key TRICKTOR to c_1984
Showing the first lines of the encrypted book
MZBNO:      GWEXKMGX XWXAKG-HYNF
RNKPQB:     ZSFKXM QBPSCE (GAGEWCERD WH OKWT UCIKB) (1903-1950)
IOIM FVG
MAOGMVZ 1

KD POJ T SZKQAH THCL FKR WE TGZKV, TBU MYM EVHQBL NMTO LHIBBQPQ MVZKKMGX.
PWELKWP CFWKA, YQU MAWE GLHBVXR ZGKW JSL PIXRAV SG OE XWNQBM HF XJKCZX HYX
MQNO PWEW, JTKZISU JLQEUEM KAIWWQA HYX XTCCL RFHIA QP OWTMFZA WTBJBFVU,
DACLZY VQD JIZVBTA OGCLZY BQ ZKSMXEB C CPWIE FN IBBHKR UCUD YFFF VVVOKWEZ
RTQXZ KZMY PKW.

EXP 2: Decrypting with key TRICKTOR to r_1984
Comparing the decrypted version with the original
No Output Means All OK

```

```
EXP 3: Encrypting with key NAVYSEALSARECOMING to c_1984
Showing the first lines of the encrypted book
GIOJW:      RIYWTVIP SUOUZL-FJSJ
EUEZOI:     KGCDOR UEWZJD (TSPMDFRAA AN RXVC WJSMR) (1903-1950)
ASRK SPS
OPNVGEM 1

GL AAD S BIMIVF KBRQ DVW AR AAJIC, EPR FPR IYOXIK AECW SKVKYUVT ZUIMRWIN.
HANJXQB EUVZU, HDQ ULIY FUQDNSP QAZB HDQ TVELKT ZR CB QNSUET OM WWCLHE KLG
JUTR CVNY, QDMPAWD HYKQWTL ZURJSYL TSW GCEUG PWBXF OA TAGTZJY DEPGUWAY,
GHJSYL NZL QLMEYXG RTBUBF LS PCWVVRV O EEVXY OA EJMTEQ DLWV TDWZ KATZPARG
LDOEK YWFP UOZ.

EXP 3: Decrypting with key NAVYSEALSARECOMING to r_1984
Comparing the decrypted version with the original
No Output Means All OK
```

Tras Encriptar, Desciframos utilizando la clave y usamos el commando **diff** con el fichero de texto original en mayusculas y el descifrado. Vemos que se realiza correctamente.

Criptoanálisis: Vigenere

Lo importante del método de Vigenere es que sin un ordenador sería muy difícil descifrar el mensaje cifrado ya que todo depende de un análisis estadístico de las frecuencias de las letras en el mensaje cifrado. De hecho utilizamos dos métodos que trabajarán juntos para confirmar la clave.
La clave para descifrar Vigenere viene en que una vez sabiendo el tamaño de la clave, se puede dividir el texto en tantas “columnas” como caracteres de la clave. Cada letra de esa columna ha sido cifrada por una misma letra de la clave ya que se repite hasta cubrir el

texto. Esto convierte la tarea en una de descifrar Cesar. Una vez hecho esto, se aplica análisis estadístico a cada columna de manera que si la letra que más aparece en la columna 1 es la 'F', hay grandes probabilidades de que la F en la primera columna se corresponda a la letra 'E' y por tanto podemos llegar a la conclusión de que se ha utilizado la letra 'B' (corresponde a 1) para cifrar esa columna. Sucesivamente por columnas, se aplica el mismo procedimiento hasta llegar a obtener la clave entera.
Los dos siguientes métodos tratan de averiguar la longitud de la clave.

Kasiski

El método de Kasiski consiste en encontrar repeticiones de segmentos de texto (por ejemplo AHG) y contar las distancias entre ellas. Si el texto es largo, y utiliza mucho la palabra "LOS", en muchos momentos coincidirá que están cifrados por la misma parte de la clave. Por tanto podemos utilizar el MCD de todas las distancias para concluir la longitud de la clave. Por supuesto, hay muchas veces que se encontrará una repetición que no corresponde a la palabra original y que es pura coincidencia. Sin embargo, para evitar que esto influya demasiado, descartamos las claves de Tamaño 1 y 2.
Es importante comentar que el método no es perfecto y muchas veces falla por pura coincidencia. Sin embargo, normalmente a pesar de fallar, sí que devuelve al menos un factor o un múltiplo de la longitud de la clave.
Para que sea más efectivo, hemos buscado varias palabras repetidas distintas, y hemos ido guardando en una tabla el mcd de cada una de ellas.

Indice de Coincidencia

El índice de coincidencia es una forma de medir lo parecido que son dos textos. Una vez más se divide el texto en columnas, aunque esta vez sucesivamente pasando por M = 1,2,3… hasta un máximo. Para romper Vigenere, la idea es tomar las frecuencias de las letras de cada columna y calcular su índice de coincidencia.
Teniendo en cuenta que el índice de coincidencia de un lenguaje aleatorio es de 0.038, y que el del Inglés es de 0.065, buscaremos índices de coincidencia cercanos a 0.065, deduciendo que para el M con que más se acerque será la longitud de la clave. También hay que tener en cuenta que para una longitud L determinada, habrá altos índices de coincidencia al dividir el texto en NL columnas (con N pequeño). Esto se tiene en cuenta y normalmente será el MCD de todos los M con el que se obtienen Indices de coincidencia altos.

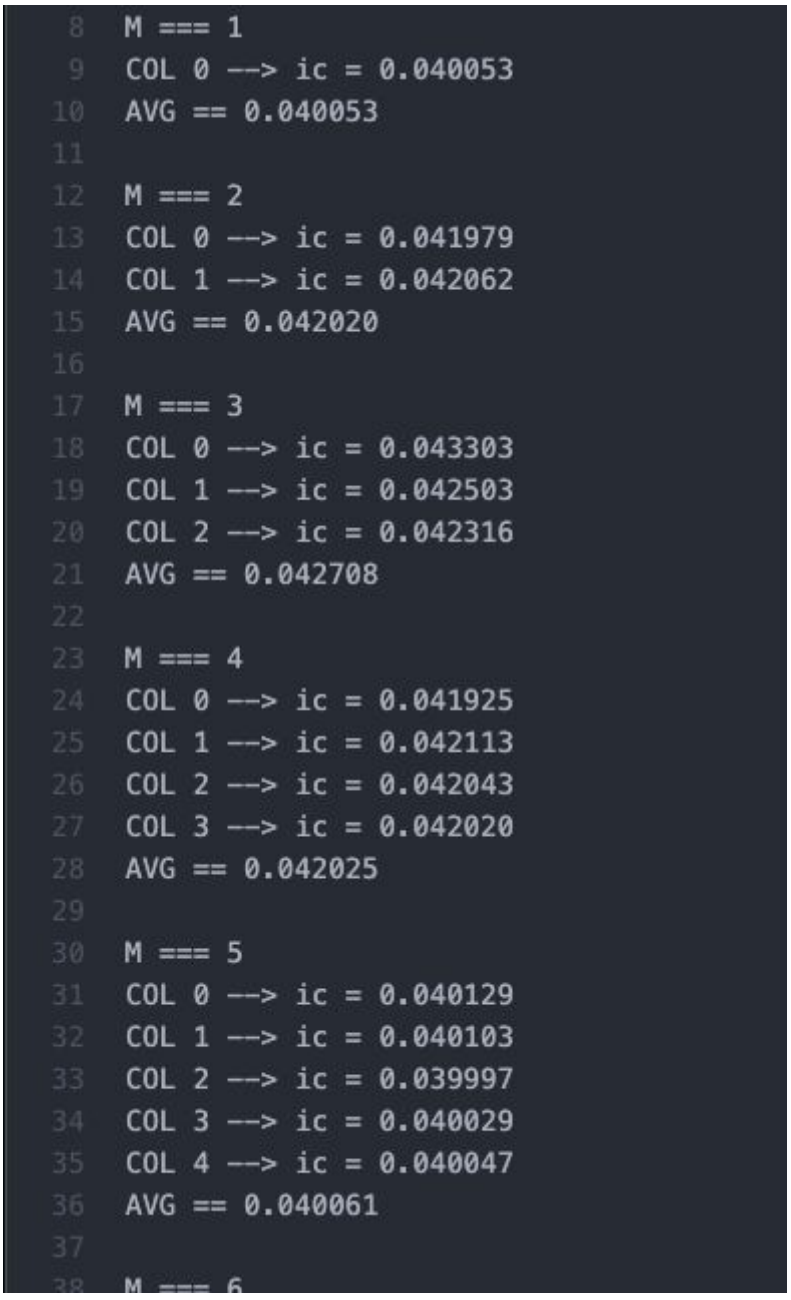
Salidas

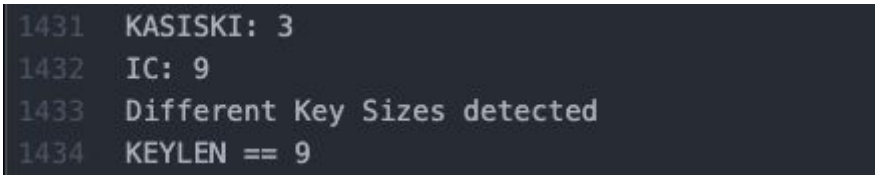
Aquí tenemos unos ejemplos de las salidas del programa Decypher.
Con Kasiski indicamos simplemente los valores de las mcds de distancias que se han ido calculando para las diversas palabras repetidas encontradas.

```
279  KEYLEN == 3
280  [ 3  9  47  9  9  101  3  4  5  210109  3  7  3  3  10  34  6  59519  6  3  6  3  6  4  10 ]
281
282  Most Probable KEYLEN: 3
```

En cambio para el Indice de Coincidencia mostramos los valores del índice de coincidencia para cada columna en cada iteración de M y luego la media de estos valores.

En caso de que haya varios valores que coinciden, se muestra un mensaje de aviso y se toma el MCD de todos.





Finalmente, el programa llama a la función “decypher_vigenere” con la longitud de clave obtenida y es este el programa que devuelve la clave.

```
570
571  KASISKI: 3
572  IC: 6
573  Different Key Sizes detected
574  KEYLEN == 6
575  PWD: WINTER
```

```
283
284  KASISKI: 3
285  IC: 18
286  Different Key Sizes detected
287  KEYLEN == 18
288  PWD: NAVYSEALSARECOMING
289
```

```
858  KASISKI: 3
859  IC: 6
860  Different Key Sizes detected
861  KEYLEN == 6
862  PWD: BOTTLE
```

Ya con esta clave, se puede proceder a descifrar utilizando el programa de Vigenere

```
37  echo '\nEXP 3: Decrypting with key NAVYSEALSARECOMING to r_1984'
38  ./v -D -k NAVYSEALSARECOMING -i c_1984.txt -o r_1984.txt
39  echo 'Comparing the decrypted version with the original'
40  diff cap_1984.txt r_1984.txt
41  echo 'No Output Means All OK'
```

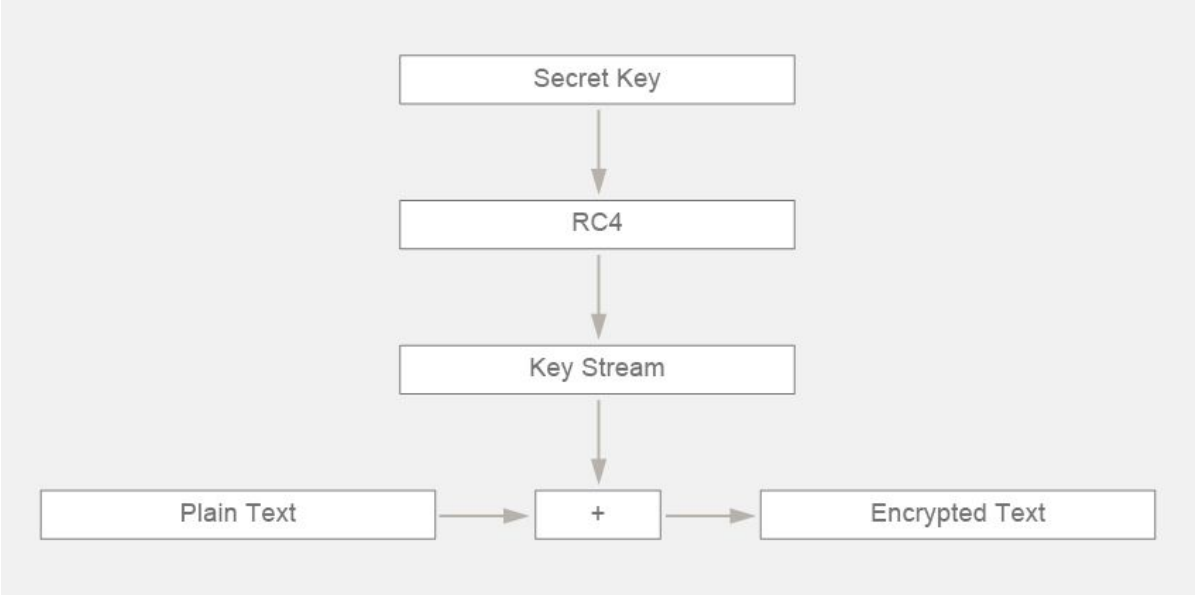
Finalmente, hemos probado con varios tamaños de clave y tamaños de texto. Para textos pequeños, es fácil que se equivoque al descifrar porque no hay mucha información y las frecuencias no son tan exactas como en un texto largo:

```
KASISKI: 3
IC: 3
KEYLEN == 3
PWD: ABR
fons@fons-mb:~/cripto_ult_BAK$ ./v -C -k ABC -i short.txt -o c_short.txt
fons@fons-mb:~/cripto_ult_BAK$
```

Esto ocurre también porque cuánto más se acerque el tamaño de clave al tamaño del texto a cifrar, más aleatoriedad hay entre las columnas (menos caracteres por columna implica que más distorsionado saldrá el índice de coincidencia).

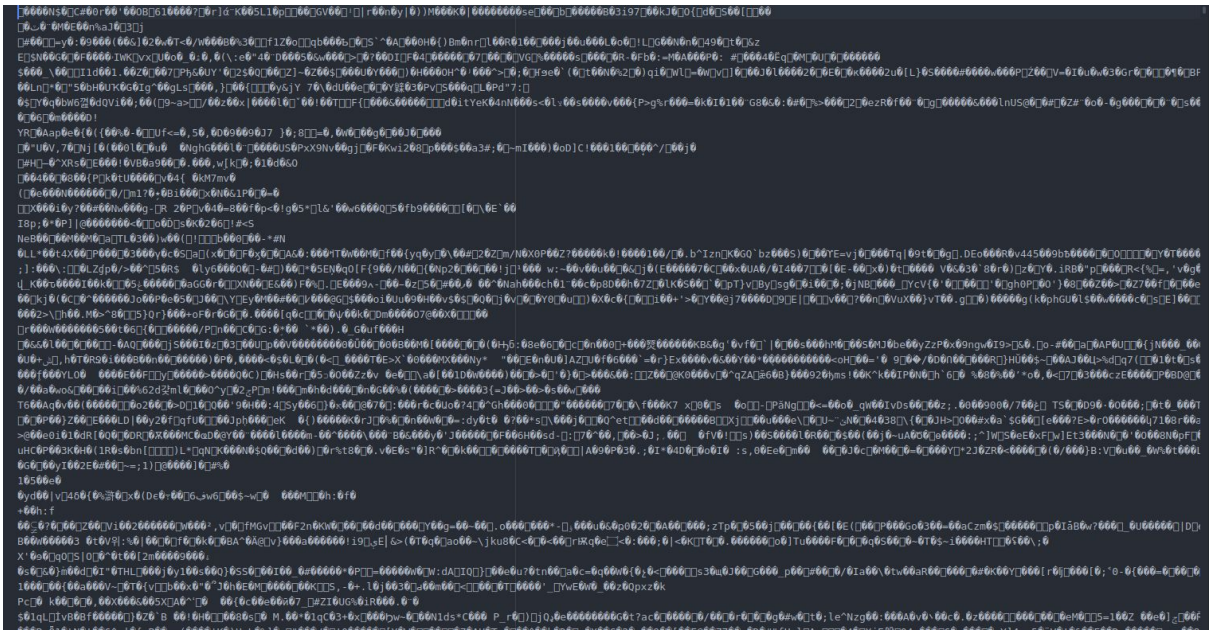
Cifrado de Flujo

Vamos a generar un cifrado de flujo para poder cifrar el libro 1984. Hemos decidido implementar RC4, que es fácil de implementar y hasta la fecha no se conoce un método para romper un texto aleatorio cifrado con este algoritmo.



En el cifrado de flujo, partiendo de una clave secreta compartida por emisor y receptor, generando así un flujo de cifrado pseudoaleatorio.
Para generar el flujo del cifrado, una permutación de todos los bytes posibles (#define N 256), es decir del 0 al 255, gracias a la función *cifradoFlujo()*. Así, se formará el vector de que se utilizará como entrada en el siguiente paso del algoritmo. Cabe destacar que para cada iteración se tiene en cuenta el resultado de la iteración anterior (es por esto que es más difícil de atacar). El resultado que obtenemos es un vector S, que es pseudoaletorio, puesto que es fácilmente replicable por alguien que tenga la key. Nos hemos creado una función auxiliar *swap* para hacer el intercambio.
Una vez creado este vector y junto con las variables i y j, se generará el flujo. En cada iteración del algoritmo se emite un byte de flujo de cifrado y se intercambian dos elementos del vector (en todo momento se depende de las variables i y j). A cada salida de éste, se le hará una XOR para generar el texto cifrado final.
Como hemos dicho anteriormente, cualquiera que sepa la key podrá descifrar el texto. Por tanto, podemos llamar al programa, con la clave con la que hemos cifrado el texto y con el fichero encriptado. Obtendremos en otro fichero el texto plano original.
Debido a lo sencilla que es la operación de combinación, el *RC4* se presta muy dado a manipulaciones. Básicamente, si un atacante invierte un bit en el texto cifrado, el receptor recogerá el bit descifrado invertido también. Mientras el atacante no conozca el texto plano, cualquier alteración del mensaje que haga no producirá nada interesante. Esto sería aceptable si no fuera porque en las aplicaciones de la vida real, tales como HTTPS, el atacante conoce gran parte del texto plano.

Adjuntamos un fragmento de la salida obtenida tras cifrar con la clave=123, 1984.txt (el resto del fragmentos se puede obtener ejecutando el programa desde una terminal):



Producto de Criptosistemas Permutación

El producto de Criptosistemas de Permutación combina dos cifrados de permutación idénticos pero el primero permuta filas y el segundo columnas.
Como claves tiene dos vectores K1, K2, que son las permutaciones y cuyas longitudes determinan la longitud de los bloques en los que se dividirá el texto.
Teniendo en cuenta que la longitud de K1 es ‘n’ y el de K2, ‘m’ podemos ver que el tamaño del bloque será m*n. Esto implica que el texto a cifrar siempre tiene que ser de longitud%m*n = 0. Sin embargo, esto no siempre ocurre y para eso es necesario el Padding. Padding lo que hace es añadir un caracter al final del texto hasta llegar a la longitud deseada.

Por ejemplo: sea nuestro texto

AAA
BBB
CCC
D

y sea el tamaño de bloque 15. Es necesario añadir unos caracteres al final para llegar a 15.

Añadiendo el carácter ‘X’, por ejemplo, se convierte el texto en: AAA BBB CCC DXX XXX
Ahora, supongamos que K1 = [1 3 2 4 0]. Esto quiere decir que la fila 0 va a escribirse en la posición 4, la 1 en la 0 etc…
Aplicando la permutación al texto de arriba, quedaría:

BBB
DXX
CCC
XXX
AAA

Ahora, supongamos que K2 = [1 2 0]. Esto quiere decir que la fila 0 va a escribirse en la posición 2, la 1 en la 0 etc…
Aplicando la permutación al texto siguiente, quedaría:

ABC		BCA
ABC		BCA
ABC	---	BCA
ABC		BCA
ABC		BCA

Es un cifrado potente porque sin conocer los vectores, ni el texto plano, queda un texto totalmente irreconocible. Además es importante comentar que aunque no cambia ninguna letra, es decir, mantiene el índice de coincidencia igual, no es vulnerable a un ataque por Indice de Coincidencia ya que no hay ninguna repetición que se pueda utilizar, nada que nos diga el tamaño de bloque.

Por comodidad, hemos tenido que eliminar todo caracter que no sea una letra mayúscula. El resultado es que el texto cifrado y descifrado queda todo pegado.
Aquí tenemos algunos ejemplos:

```
Showing the first lines of the mini text
BBB
CCC
DDD
EEE
FFF
BBB
CCC
DDD
EEE
FFF


EXP 1:
 Encrypting with perms: -k1 [ 1 3 2 4 0 ] -k2 [ 1 3 2 0 ] to c_mini
Showing the first lines of the encrypted text
CDDCFBFFEEEDBCCBBCBBEFEEXXXXFXXFXXXXDDDC

EXP 1:
 Decrypting with perms: -k1 [ 1 3 2 4 0 ] -k2 [ 1 3 2 0 ] r_mini
Showing the first lines of the decrypted text
BBBCCCDDDEEEFFFBBBCCCDDDEEEFFFXXXXXXXXXX
Differences between original text (padded) and deciphered:

No Output Means All OK
```

```
Showing the first lines of the 1984 text
Title:      Nineteen eighty-four
Author:     George Orwell (pseudonym of Eric Blair) (1903-1950)
PART ONE
Chapter 1

It was a bright cold day in April, and the clocks were striking thirteen.
Winston Smith, his chin nuzzled into his breast in an effort to escape the
vile wind, slipped quickly through the glass doors of Victory Mansions,
though not quickly enough to prevent a swirl of gritty dust from entering
along with him.
Encrypting with perms: -k1 [ 7 2 0 1 3 5 4 6 ] -k2 [ 4 0 2 3 1 ] to c_1984
Showing the first line of the encrypted text
LOWERIENEEETTLITNNEIFGTYHGTORHUORAUEERGOSIWATIOERFULSEPMDNYOICLABHOECNTRARPRATEPRRSTEIDAYDGARIBLHCOT

EXP 2:
 Decrypting with perms: -k1 [ 7 2 0 1 3 5 4 6 ] -k2 [ 4 0 2 3 1 ] r_1984
Showing the first lines of the decrypted text
TITLENINETEENEIGHTYFOURAUTHORGEORGEORWELLPSEUDONYMOFERICBLAIRPARTONECHAPTERITWASABRIGHTCOLDDAYINAPRI
Differences between original text (padded) and deciphered:

No Output Means All OK
```

Descifrar es tan sencillo como en el caso de Vigenere. Solo es necesario invertir los vectores K1, K2 de manera que mapeen cada fila/columna a su lugar original, y llamar a las funciones de permutar.

Por ejemplo en el caso anterior con el vector K2 = [1 2 0], tenemos que K2* = [2 0 1].
De esta manera, aplicando las permutaciones igual que antes

BCA		ABC
BCA		ABC
BCA	---	ABC
BCA		ABC
BCA		ABC

Aquí un par de ejemplos descifrados.

En cuanto a criptoanálisis posible, tendríamos primero que averiguar el tamaño de los bloques y luego probar con distintos tamaños de los vectores. Porque si el tamaño del bloque es 15, puede ser que la longitud de K1 = 5 y K2 = 3 o vice versa. Probando para cada una de estas longitudes, habría que luego probar todas las posibles combinaciones de permutaciones posibles.
Para longitud K1 = n y longitud de K2 = m tendríamos $2 \times Factores(nm) \times (n! \times m!)$ posibilidades, y tendríamos que ir comprobando manualmente si alguna deja el texto en claro. Para números pequeños como 5 y 3 tenemos 1440 posibilidades. Y esto es teniendo en cuenta que hemos dado con el tamaño del bloque. Para n = 10 y m = 15 tendríamos $2 \times 6 \times (10! \times 15!) = 56943464959180800000$ posibilidades. Es decir, no es viable descifrar el código, sin tener algo de ayuda.
Aquí es donde entra la posibilidad (como en el apartado 5) de tener la máquina y el texto cifrado. Con la máquina, podemos ir probando tamaños de bloque hasta ver que no se ve modificada la longitud del bloque probado y por tanto habríamos dado con la longitud de bloque. Luego, (como en los ejemplos de antes) tomamos factores de la longitud del bloque y hacemos filas con ellos hasta ver que una fila se cifra entera (como AAAAA en los

ejemplos anteriores). De esta forma ya sabríamos la longitud de los vectores. Una vez sabiendo la longitud de los dos vectores, solo quedaría tomar un bloque de texto de manera que todas las filas o columnas sean iguales para obtener las permutaciones de columnas o filas.
Habiendo obtenido las permutaciones, es fácil invertirlas y descifrar el texto cifrado. Pero, solo es posible teniendo la máquina.s