Cifrado parcial (Análisis de Código). Reporte 2.

Marcos Daniel Calderón Calderón
Maestría en Ciencias de la Computación
Centro de Investigación en Matemáticas (CIMAT)
Guanajuato , Gto.
marcos.calderon@cimat.mx

***Resumen*—En este documento se explica de manera detallada el código de cifrado parcial.**

I. Explicación de código.

El código de cifrado parcial está formado de los siguientes pasos.

1. Se utilizan 4 mapas caóticos. Para la formación de estos mapas caóticos, se necesita lo siguiente:
   - Un arreglo de tamaño 4 para almecenar los pesos no enteros (**float W[NN]**).
   - Un arreglo de tamaño 4 para almacenar los parámetros no enteros y enteros respectivamente (**float be[NN]**, **unsigned int beta[NN]**).
   - Un arreglo de tamaño 4 para almacenar la clave no entera (**float Key[NN]**).
   - Arreglos de tamaño 4 para almacenar los pesos y las condiciones iniciales enteras (**unsigned int W_int[NN+1]**, **unsigned int IC_int[NN]**).
2. Ahora, el siguiente paso fué inicializar los pesos enteros y no enteros, las condiciones iniciales, la clave, y los parámetros $beta$, a continuación se muestra la inicialización.

Cuadro I. Inicializaciones de mapas caoticos.

W[i]	W_int[i]	IC[i]	Key[i]	be[i]
0.000000	4294967295	0.480000	0.480000	2.000000
0.250000	1073741824	0.530000	0.530000	3.000000
0.500000	2147483648	0.310000	0.310000	4.000000
0.750000	3221225472	0.680000	0.680000	5.000000

3. Una vez que hemos hecho las inicializaciones anteriores, ahora, lo que se hace es ejecutar un ciclo 7 veces ($NumIt_i = 7$), en cada iteración, se calcular un valor para h con los valores anteriores de los mapas caóticos, y después, se calculan nuevos valores para los mapas caóticos, se utiliza el siguiente código:

```
for(j=0; j<NumIt_i; j++)
{
h=0;
for(k = 0; k<NN; k++)
h+= W[k]*IC[k];
for(i=0; i<NN; i++)
IC[i] = (1.0 -ep)*Renyi(IC[i],be[i]) + ep*h;
}
```

Resultado obtenido para la última iteración en este fragmento de código:

Cuadro II. Resultado última iteración.

IC[i]
131.139221
181.190384
334.825470
279.543152
383.958374
549.630310
1024.313232

4. Ahora, el siguiente paso consiste en inicializar los mapas caóticos enteros, esto se hace mediante una operación de casting sobre los valores obtenidos en el paso anterior para los macas caóticos no enteros, también, se hace la inicialización de la clave, en este caso, la clave toma el mismo valore que los valores iniciales para los mapas caóticos enteros. A continuación se muestra el código y los valores obtenidos.

```
for(i=0; i<NN; i++)
{
IC_int[i] = (unsigned int) IC[i];
Key[i] = IC_int[i];
}
```

Cuadro III. Inicializaciones de mapa entero y clave.

IC_int[i]	Key[i]
524	524.000000
702	702.000000
1043	1043.000000
1926	1926.000000

5. Ahora, es necesario leer los archivos que contienen los datos que vamos a cifrar. Para lograr esta tarea, se utiliza la función **fseek** para posicionarnos en el último carácter del archivo leído (final del archivo), con la orden **SEEK_END**, la operación anterior se ejecuta al desplazar la posición actual de lectura o escritural punto indicado, en este caso, indicamos el final der archivo. La función devuelve un cero si no ha habido problemas, también, se necesita un parámetro que indica el desplazamiento que se ha de realizar. a partir de la referencia indicada, en este caso, se indica que no ocurre ningún desplazamiento, nos quedamos en la posición final indicada. También se utiliza la función **ftell**, para archivos binarios se devuelve el número de bytes desde el inicio hasta donde nos posicionamos en el archivo (el final). También, nos volvemos a

posicionar en el inicio del archivo con la instrucción **rewind**.

6. Almacenamos memoria para guardar la información leída, se guarda un espacio en memoria dinámica del tamaño del número de bytes leídos.
7. Como siguiente paso, es necesario declarar todas las variables que vamos a necesitar. A continuación, se da una lista de todos los valores que vamos a utilizar en el programa.
   - $Pack\_Size = 400$. Este es el tamaño del paquete RTP recibido.
   - $BlockSize = 8$. Este es el tamaño de un bloque, todavía no se en que se utiliza.
   - $pos\_f = 0$.
   - $Num\_Blocks = 20$.
   - $Num\_bits\_encry = 32$. Este número de bits se utiliza para cifrar.
   - $Bl\_Size\_byte = Pack_Size/Num\_Blocks$. Esto es la operaciónÑ $\frac{400}{20} = 20$ bytes.
   - $Bl\_Size\_bit = Bl\_Size\_byte*8 = 160$. Número de bits.
   - $Num\_It = Bl_Size\_bit/Num\_bits\_encry$.
   - $rand\_Bl, temp$.
   - pos_byte.
   - $unsignedintNum\_pack = lSize/400$.
   - $char * Pack\_temp$.
   - $Pack\_temp = Dat$.
8. También declaramos una variable de tipo apuntador **Pack_temp** que apunta al mismo arreglo que **Dat**.
9. Declaramos un arreglo que tenga de casillas el número de bloques que conformarán a cada paquete RTP. **unsigned int Blocks[Num_Blocks]**.
10. Ahora, declaramos un arreglo de carácteres de tamaño cuatro y los inicializamos con las primeras letras minúsculas del alfabeto. El resultado es el siguiente:

```
char PText[(Num_bits_encry/8)];
//initialize Plain Text
for(i=0; i<(Num_bits_encry/8); i++){
PText[i] = (char)'a'+i;
}
-- a ---- b ---- c ---- d
```

11. Ahora, el arreglo de nombre **Blocks** (de tamaño 20) se inicializa con los siguientes valores:

```
0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
```

12. Ahora, el siguiente paso es tomar el tiempo de ejecución del algoritmo.
13. Una vez que han ocurrido todos los pasos anteriores, ahora es necesario ejecutar el algoritmo. Para cada uno de los $Num\_pack = 2547$ paquetes RTP, se hace lo siguiente:
    a) Inicializamos el arreglo Blocks de la siguiente manera:

```
0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
```

    b) Ahora, recorremos los bloques como indica el paper, lo que queremos es hacer el proceso de intercambio de bloques de un paquete RTP. Por lo tanto, este proceso comienza con la siguiente instrucción:

```
for(i=Num_Blocks; i>0; i--){//codigo}
```

II. Observaciones de código.

A continuación, se mencionan algunas observaciones que pueden ayudar a la optimización de código:

1. En el siguiente fragmento de código:

```
rand_Bl = IC_int[0]%i;
//shuffling
temp = Blocks[i-1];
Blocks[i-1] = Blocks[rand_Bl];
Blocks[rand_Bl] = temp;
```

   puede ocurrir que $rand_Bl = (i - 1)$, por lo tanto se hace un intercambio innecesario, y esto ocurre frecuentemente, como se puede observar en los siguientes resultados:

```
aleatorio: 15     i-1:   15
aleatorio: 14     i-1:   14
aleatorio: 0     i-1:   0
aleatorio: 5     i-1:   5
```

   los resultados anteriores se obtuvieron al realizar 24 iteraciones del código mencionado, obtener cuatro coincidencias es un porcentaje alto, tomando en cuenta que ocurren miles de iteraciones para un archivo.
2. EL siguiente paso es conocer la direccion endonde estamos ubicados, esto se hace al indicar el bloque que se va a intercambiar por el tamaño del bloque (**pos_byte = Blocks[i-1]*Bl_Size_byte**).
3. Ahora, se aplica un ciclo 5 veces, que es lo que vale la variable **Num_It**, este valor se obtuvo al dividir lo siguiente: $\frac{Bl\_Size\_bit}{Num\_bits\_encry}$. El numerador vale 160 (es el tamaño del bloque en bits)y el denominador vale 32. Adentro de ese ciclo, ocurren los siguientes proceso.
   - Primero, se calcula un valor de h con los valores anteriores de los cuatro mapas caóticos.
   - Ahora, con el valor de h calculado en el paso anterior, se calculan nuevos valores para los mapas caóticos Renyi.
   - Ahora, se enuentran los carácteres que se encuentran en el texto original en la posición **pos_byte** y en los siguientes 3 lugares, recordemos que estos 4 carácteres se guardan en el arreglo **PText**.
   - Ahora, ocurre el proceso de inversión de bits para cada bloque que se va a intercambiar, esto se hace por medio de un ciclo, donde la principal instrucción de terminación se puede ver abajo, también debemos recordar que $BlockSize = 8$ y además $Num\_bits\_encry = 32$. Cada vez que se ejecuta el ciclo, **pos_f** vale cero inicialmente, y se va incrementando de acuerdo al valor que tome **p**: un valor aleatorio entre 0 y 7.

     ```
     while(pos_f + BlockSize <
     ```

```
Num_bits_encry)
```

Cuadro IV. ALGUNOS VALORES DE POS_F.

pos_f
0
7
8
11
18
21