Reporte 8. Cómo utilizar de manera eficiente todos los bits de los mapas caóticos.

En reportes anteriores hemos utilizado ejemplos donde se utilizan mapas caóticos para el cifrado de archivos. Sin embargo, no era óptima la manera de utilizarlos: no se utilizaban todos los octetos de bits. En este reporte se especifica una manera adecuada para utilizar todos los bits.

Líneas de código utilizadas para utilizar todos los octetos de bits.

Se han cambiado algunas líneas del programa de cifrado parcial para utilizar todos los octetos de bits, a continuación, se presentan las más importantes:

/*Si un numero aleatorio es impar...*/

```
if(p&1){
        posicionInversion = bitRef+
 ( ((Xn[count]&mascaras[countMask])>>shift[countMask]) % (BF-p));
}
                    /*Si el un numero aleatorio es par...*/
else{
        posicionInversion = bitRef-( ((Xn[count]&mascaras[countMask])>>shift[countMask]) %
p);
}
```

A continuación detallamos algunos aspectos importantes del código anterior:

1. **Xn[count]** es el arreglo de mapas, donde count tomará un valor entre 0 y 4.
2. **mascaras[countMask]** es un arreglo que contiene los valores necesarios que se aplicarán como máscara para obtener cada uno de los cuatro octetos de un mapa caótico de 32 bits. A continuación mostramos unos ejemplos de los valores que tiene este arreglo:

   ...

   255

   65280

   16711680

   4278190080

   255

   65280

   16711680

   4278190080

...

3. **shift[countMask]** contiene los valores de desplazamiento de bits a la derecha que se deben de aplicara para cada octeto y así obtener el valor de los ocho octetos elegidos. Los valores que tiene este arreglos son los siguientes:

   ...
   0
   8
   16
   24
   0
   8
   16
   24
   0
   8
   16
   24
   ...

4. Todos los arreglos mencionados **tienen un tamaño de 256.** Esto es bueno porque dicho número se ajusta de manera adecuada al número de mapas y al número de octetos que forman al mapa. Además se llenana de manera cíclica: significa que los datos se repiten cuando ya se ha completado un ciclo.

¿Cómo decidir en qué momento cambiamos de mapa?

Para decidir en qué momento cambiamos de mapa, utilizamos las siguientes instrucciones:

```
count+=(countMask&3)/3;
countMask++;
```

countMask es un contador cuyos valores oscilan entre 0 y 255, éstos valores se utilizan para recorrer los arreglos mencionados anteriormente. A continuación mostramos algunos valores que se obtienen:

- 247	- 252	- 1	- 6
- 248	- 253	- 2	- 7
- 249	- 254	- 3	- 8
- 250	- 255	- 4	- 9
- 251	- 0	- 5	- 10

La máscara aplicada nos devuelve valores entre 0 y 3. La división entera nos sirve para sumar un valor al contador en el momento adecuado: cuando el resultado de aplicar la máscara es 3, al dividir entre tres, obtenemos un valor de 1, y **el contador aumenta en 1 y por lo tanto, pasa al siguiente mapa caótico.**

Algunos valores de p obtenidos.

En el código que se muestra al inicio, se puede ver que es necesario obtener un valor de p. Dicho valor se calcula de la siguiente manera (todavía estamos en proceso de optimización de código:

```
p=(  ((Xn[count]&mascaras[countMask])>>shift[countMask]) %(BF-2)  ) + 1;
printf("\n Valor de p: %u", p);
count+=(countMask&3)/3;
countMask++;
```

A continuación se pueden ver algunos valores obtenidos para p, y éste es el valor que se compara si es par o es impar.

```
Valor de p: 6
Valor de p: 3
Valor de p: 5
Valor de p: 5
Valor de p: 5
Valor de p: 6
Valor de p: 6
Valor de p: 5
Valor de p: 3
Valor de p: 2
Valor de p: 3
Valor de p: 5
Valor de p: 6
Valor de p: 3
Valor de p: 5
```

Valores que toma count

La variable **count** nos indica el mapa caótico que se va a utilizar para generar números pseudoaleatorios. **De acuerdo al algoritmo presentado, se supone que el valor de count estará fijo en cuatro veces, esto es así porque vamos a tomar cada uno de los cuatro octetos del mapa caótico representado por count.** A continuación, mostramos que esto ocurre, se muestra una lista de algunos valores para la variable count, los valores con un guión pertenecen al los números pseudoaleatorios utilizados para el proceso de intercambio de segmentos. Los resultados nos muestran que no importa que se pase del proceso de inversión de bits al proceso de intercambio: si falta algún octeto a procesar, se procesa, nada se desperdicia. Ejemplos:

2	-- 3	0
2	-- 3	0
2	-- 3	0
2	-- 3	0
3	-- 0	1
3	-- 0	1
3	0	1
-- 3	0	1
-- 0	1	2
-- 0	1	2
-- 0	1	2
-- 0	1	2

Algunos resultados.

A continuación, mostramos algunos resultados, primero mostramos los arreglos que nos indican la manera en cómo se intercambiarán los segmentos de los paquetes RTP. Para el ejemplo mostrado aquí, se utilizaron dos paquetes RTP:

Intercambio del Primer paquete RTP:

2 8 5 1 7 6 3 9 4 0

Intercambio del Segundo paquete RTP:

4 5 6 9 3 2 7 8 0 1

Ahora, mostramos los datos originales y el resultado, respectivamente:

Datos originales:

255 216 255 224 0 16 74 70 73 70 0 1 1 1 1 43 1 43 0 0 255 219 0 67 0 5 3 4 4 4 3 5 4 4 4 5 5 5 6 7 12 8 7 7 7 7 15 11 11 9

```
12 17 15 18 18   17 15 17 17 19   22 28 23 19 20   26 21 17 17 24   33 24 26 29 29
31 31 31 19 23   34 36 34 30 36   28 30 31 30 255   192 0 11 8 0   60 0 60 1 1
```

Resultado:

```
34 18 41 146 9   30 26 131 37 21   129 135 37 30 20   128 90 4 65 82   1 33 129 44
127   66 38 21 132 65   59 131 35 96 33   135 141 4 131 41   251 154 36 103 132
231 124 252 228 18
```

```
97 154 10 76 141   23 63 152 195 159   96 0 170 62 160   44 35 62 4 1   102 29 80
19 191   50 16 159 131 28   217 35 61 94 123   200 64 170 0 130   8 63 240 208 26
141 7 48 17 18
```