IMPLEMENTACIÓN DEL ALGORITMO A.E.S. (ADVANCED ENCRYPTION STANDARD)

Patricia V. Maizel

Directora: Mónica Liberatori.

Laboratorio de Alta Frecuencia.

Departamento de Electrónica.

2004

INDICE

INDICE 2
1.Resumen 3
SECCIÓN II 5
INTRODUCCIÓN 5
2.Introducción 6
Principios de la criptografía 6
¿Qué es la criptografía? 6
SECCIÓN III 8
DESCRIPCIÓN DEL ALGORITMO A.E.S. 8
3.Algoritmo A.E.S-Rijndael 9
3.1.Transformación ByteSub 11
3.2.Transformación ShiftRow 12
3.3.Transformación MixColumns 12
3.4.Expansión de la clave 14
3.5.Proceso de desencriptado 18
3.5.1.Transformación Inv-ShiftRow 18
3.5.2.Transformación Inv-ByteSub 19
3.5.3.Transformación Inv-MixColumn 21
SECCIÓN IV 25
DESCRIPCIÓN DEL SOFTWARE 25
4.Descripción del Software 26
SECCIÓN V 40
MANUAL DE USO 40
5.Manual de uso 41
5.1.Sobre el presente manual 41
5.1.1.Objetivo 41
5.1.2.Restricciones 41
5.2.Interfaz gráfica 41
5.2.1.Panel de encriptado 42
5.2.2.Panel de desencriptado 45
SECCIÓN VI 49
MEDICIONES DE VELOCIDADES DE ENCRIPTADO Y DESENCRIPTADO 49
6.Medición de velocidades de encriptado y desencriptado 50
SECCIÓN VII 55
CONCLUSIONES 55
7.Conclusiones 56
SECCIÓN VIII 58
BIBLIOGRAFÍA 58
8.Bibliografía 59
SECCIÓN IX 60
ANEXO A 60
CÓDIGO FUENTE DEL PROGRAMA CORRESPONDIENTE A LAS FUNCIONES DE ENCRIPTADO Y DESENCRIPTADO 60
SECCIÓN X 132
ANEXO B 132
CÓDIGO FUENTE DEL PROGRAMA CORRESPONDIENTE A LA INTERFAZ GRÁFICA 132

Resumen

El presente trabajo describe el funcionamiento y los detalles de implementación en software del algoritmo de encriptado Rijndael, recientemente elegido como estándar A.E.S.(Advanced Encryption Standard). Se propuso implementar un código apropiado para encriptado/ desencriptado Rijndael de tal manera de concentrar el esfuerzo en la obtención de una interfaz sencilla con soporte para procesamiento rápido y portabilidad. Con este propósito se eligió el tamaño de bloque de procesamiento apropiado, pudiéndose elegir entre varias opciones, tanto para los datos a encriptar como para la clave a utilizar. Por motivos de velocidad de procesamiento, también se investigó como implementar las funciones de encriptado y desencriptado en el mismo código o por separado y cómo se generaron las subclaves necesarias para cada round. El propósito de aumentar la velocidad condujo a la indagación de la posibilidad de efectuar operaciones en paralelo. El cifrador Rijndael resultó ser el finalista elegido entre cinco candidatos para la implementación del algoritmo de encriptamiento simétrico estándar, AES, para protección de información sensible, en reemplazo del DES a partir del 26 de mayo del 2002. Se trata de un cifrador tipo bloque que puede usar claves de 128, 192 o 256 bits de longitud. Varias operaciones se definen a nivel de bytes, que representan elementos en el campo finito GF(2^8). Otras operaciones se definen en términos de palabras de 4 bytes, que se tratan como polinomios de grado cuatro y coeficientes en GF(2^8). El A.E.S. (Advanced Encryption Standard), será probablemente el algoritmo de encriptación más utilizado en el presente siglo. El crecimiento explosivo de la informática y la convergencia entre ordenadores y comunicaciones, hacen pensar que los algoritmos de encriptación corresponderán al código ejecutable de más aplicación en el futuro. Como se dijo anteriormente, el algoritmo A.E.S. encripta bloques de 128, 192, o 256 bits usando claves de 128, 192 ó 256 bits. El proceso consiste en una serie de cuatro transformaciones matemáticas, las cuales se repiten 10, 12 o 14 veces, dependiendo de la longitud del bloque y de la longitud de la clave. Las sucesivas transformaciones se pueden imaginar como aplicadas a una matriz de cuatro filas por cuatro columnas. Los sucesivos elementos de la matriz, son los 16 caracteres del texto del bloque de 128 bits a ser encriptado. Este valor para el tamaño del bloque corresponde al caso de elegir trabajar con una clave de 128 bits. En el caso de utilizar 192 bits para la clave, la matriz constaría de dos columnas más y en el caso de utilizar 256 bits habría cuatro columnas más. Uno de los procesamientos más importantes en el desarrollo del algoritmo es lo que se denomina **Expansión de la clave**. La misma consiste en la generación de diez claves distintas, ya que el proceso de encriptado mencionado previamente se repetirá diez veces o, como se describe en el algoritmo, a lo largo de diez round. De esta manera se genera una clave diferente para cada round.

En el proceso de desencriptado, se utilizan las transformaciones inversas a las utilizadas en el proceso de encriptado, y la primer clave a emplear en el primer round, es la última clave generada en la **Expansión de la clave**.

El software fue realizado en C++Builder, por ser el lenguaje C uno de los lenguajes más potentes y poseer más capacidad en el manejo de memoria y de interrupciones que otros programas. Se lo puede dividir en dos partes:

Un archivo de extensión.h, que contiene todas las funciones de encriptado y desencriptado a utilizar. Los llamados de las funciones de encriptado y desencriptado se encuentran agrupados a su vez, en dos funciones denominadas **EncriptarVector** y **Desencriptar Vector**, que realizan los procesos de encriptado y desencriptado, respectivamente.

El otro archivo, es un archivo de extensión.cpp, que contiene en su cabecera el llamado del archivo de extensión.h, mencionado anteriormente, y la programación de la interfaz gráfica, donde en la programación del botón **Encriptar archivo** y del botón **Desencriptar archivo**, se encuentran los llamados a las funciones **EncriptarVector** y **DesencriptarVector**, respectivamente. La interfaz gráfica puede dividirse en dos partes, una correspondiente al panel de encriptado y la otra al panel de desencriptado, en los cuales se encuentran los botones **Encriptar archivo** y **Desencriptar archivo**, respectivamente. En la cabecera del archivo encriptado, también figuran encriptados el tamaño del archivo a encriptar, el tamaño y nombre de la extensión, para que el usuario pueda decodificarlos y desencriptar al archivo con la misma extensión original.

Respecto a las mediciones de velocidades de encriptado y desencriptado que se realizaron en el software, se puede decir que la calidad del sistema es óptima en velocidad en los diferentes sistemas operativos, y velocidades de microprocesadores utilizados en las distintas pruebas. También se pudo comprobar que al comparar dichas velocidades promedio con otras implementaciones del A.E.S., en estas últimas, las velocidades fueron más rápidas, pero hay que tener en cuenta, que la velocidad depende tanto del algoritmo, como de la implementación, el lenguaje de programación, etc. De hecho se pudo comprobar que el software realizado varía en velocidad en la misma máquina con diferentes sistemas operativos.

SECCIÓN II

INTRODUCCIÓN

2.Introducción

Principios de la criptografía

Cuando un ordenador procesa información, la misma se representa siempre con números binarios, o sea, secuencias de unos y ceros. Este tipo de representación de información se llama digital. El mundo en el cual vivimos, la información es analógica, o sea puede tomar cualquier valor. Calor, sonidos, colores, todos son datos analógicos. Los ordenadores tienen dispositivos de entrada-salida para traducir la información analógica a digital o viceversa. Lo importante es que una vez convertido en formato binario, toda información se representa internamente como números binarios y se puede procesar, transferir o encriptar sin importar su significado. Por eso todos los principios de la criptografía se aplican a cualquier tipo de información.

La criptografía tiene varios objetivos: el secreto, la integridad de los datos, y la autenticación del remitente de los datos. Entonces:

¿Qué es la criptografía?

La criptografía es un conjunto de técnicas que intentan hacer inaccesible la información a personas no autorizadas. Por lo general, la criptografía se basa en una clave, sin la cual la información no puede ser descifrada.

La tecnología de clave proporciona servicios de encriptado que aseguran las transmisiones en la red en entornos abiertos. Hay dos tipos de tecnología de claves: clave privada (encriptado simétrico), y clave pública (encriptado asimétrico).

El propósito de cualquier sistema de encriptado es asegurar la comunicación privada, pero debido al crecimiento de las redes internacionales, unido a la explosión sucedida en la tecnología de las comunicaciones, conduce a que cada vez se requieren mayores medidas de seguridad para preservar la información.

Con el uso de analizadores de protocolo con el objetivo de realizar inspecciones del tráfico de la red, se ha evidenciado el hecho que las transmisiones de datos no son seguras puesto que cualquier persona con un dispositivo semejante podría visualizar en la red los flujos de datos seleccionados. Esta situación pone de manifiesto uno de los motivos para volcarse hacia las nuevas tecnologías de encriptado.

A los métodos de encriptado de clave privada se los denomina códigos simétricos y consisten en encriptar la información con una clave que tanto el emisor como el receptor conocen y mantienen en privado. Una vez encriptado el mensaje, es ilegible y puede ser

transmitido por cualquier medio de manera segura. Este sistema da por supuesto que el intercambio de clave secreta ha sido realizado por algún medio seguro, pudiéndose aplicar con este objetivo métodos de clave pública. Una de las principales ventajas del encriptado simétrico es que las implementaciones de los algoritmos típicos resultan en altas velocidades de procesamiento en comparación con aquellas de clave pública.

Los métodos de encriptado de clave pública o códigos asimétricos consisten en la creación de dos claves relacionadas para cada usuario. Una se mantendrá en secreto y la otra es de conocimiento público. Cuando un usuario desea enviar un mensaje confidencial a otro usuario, encripta al mensaje con la clave pública del receptor y luego el receptor lo desencripta utilizando su clave privada. Los mensajes encriptados con una clave pública, sólo pueden ser desencriptados con una clave privada.

El A.E.S. es un algoritmo de encriptado de clave privada o código simétrico. Puede utilizar claves de 128, 192 o 256 bits. Procesa bloques de mayor longitud (128 bits) que cualquier otro algoritmo de clave simétrica conocido y es mucho más rápido que el D.E.S., el algoritmo previo de encriptado estándar para el uso del gobierno de los Estados Unidos con aplicación sobre datos no críticos para la seguridad nacional.

Un ataque que funciona contra todos los algoritmos de encriptado es el llamado ataque de fuerza bruta. Consiste en probar todas las claves posibles, una después de otra. El tamaño de la clave es una defensa contra un ataque de fuerza bruta, 128 bits son una defensa suficiente contra ese tipo de ataque únicamente.

SECCIÓN III

DESCRIPCIÓN DEL ALGORITMO A.E.S.

3.Algoritmo A.E.S-Rijndael

Como se dijo anteriormente, el algoritmo A.E.S. encripta bloques de 128, 192, o 256 bits usando claves de 128, 192 ó 256 bits. El proceso consiste en una serie de cuatro transformaciones matemáticas, las cuales se repiten 10, 12 o 14 veces, dependiendo de la longitud del bloque y de la longitud de la clave. Todos los ciclos, excepto el último son similares y consisten de las siguientes transformaciones:

Transformación ByteSub (Sustitución de bytes).

Transformación ShiftRow (Desplazamiento de filas).

Transformación MixColumns (Multiplicación de columnas).

Transformación AddRoundKey (Se aplica una or-exclusiva entre los bits del texto y la llave).

En el último ciclo sólo se ejecutan las siguientes transformaciones:

Transformación ByteSub.

Transformación ShiftRow.

Transformación AddRoundKey.

La primer transformación que se ejecuta es la AddRoundKey o suma EXOR entre el texto plano (sin encriptar) y la llave.

El diagrama de flujo de la Fig. 1 representa la secuencia de transformaciones mencionada.

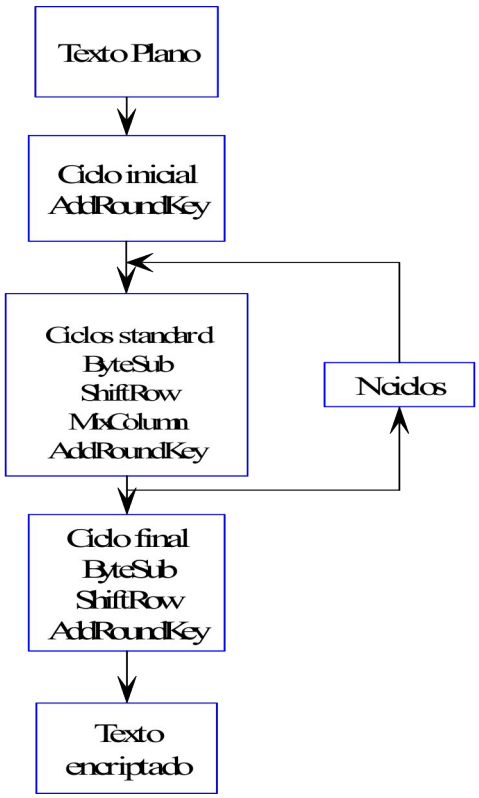

Figura 1. Proceso de encriptado

Las sucesivas transformaciones se pueden imaginar como aplicadas a una matriz A de cuatro filas por cuatro columnas, representada en la Ec. 1.

$$A=\begin{matrix} A_{00} & A_{10} & A_{20} & A_{30} \\ A_{01} & A_{11} & A_{21} & A_{31} \\ A_{02} & A_{12} & A_{22} & A_{32} \\ A_{03} & A_{13} & A_{23} & A_{33} \end{matrix} \qquad \text{Ec. (1)}$$

Los sucesivos A_{ij} son los 16 caracteres del texto del bloque de 128 bits a ser encriptado. Este valor para el tamaño del bloque corresponde al caso de elegir trabajar con una clave de 128 bits.

En el caso de utilizar 192 bits para la clave, la matriz representada en la Ec. 1 constaría de dos columnas más y en el caso de utilizar 256 bits habría cuatro columnas más.

Todas las transformaciones son aplicadas a la matriz A, excepto la transformación AddRoundKey.

Uno de los procesamientos más importantes en el desarrollo del algoritmo es lo que se denomina Expansión de la clave.

3.1.Transformación ByteSub

La transformación ByteSub, es una sustitución no lineal, que opera independientemente en cada byte de la matriz. Cada byte del bloque de entrada es invertido sobre GF (2^8) y luego sufre una transformación afín. La operación completa puede realizarse mediante una matriz de sustitución, conocida con el nombre de S-Box.

El resultado final de esta transformación de dos etapas se presenta en la Tabla 1. Esta tabla representa la S-box en formato hexadecimal.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Tabla 1 S-box

La transformación ByteSub por medio del uso de la Tabla 1 o S-box, consiste en tomar el elemento(byte) de la matriz A de cuatro por cuatro en formato hexadecimal, dicho número se transforma en el subíndice necesario para extraer un valor de la S-box. Por ejemplo, si el número de la matriz A es {12} en formato hexadecimal, 1 representa el número de fila de la S-box (columna x de la Tabla 1) y 2 el número de columna (columna y de la Tabla 1), con lo cual dicho elemento es reemplazado por {c9} en el mismo formato.

3.2.Transformación ShiftRow

A continuación de ByteSub, se aplica la transformación ShiftRow. En la misma, los bytes de las tres últimas filas son desplazados cíclicamente en distintas cantidades o posiciones (offsets). En la primer fila el offset es nulo, es decir, que la fila no se desplaza. En la segunda fila, el desplazamiento es de 1 byte hacia la izquierda, en la tercera es de 2 bytes y en la última es de 3 bytes, también hacia la izquierda.

En la Fig. 2 se ilustra gráficamente la operación mencionada.

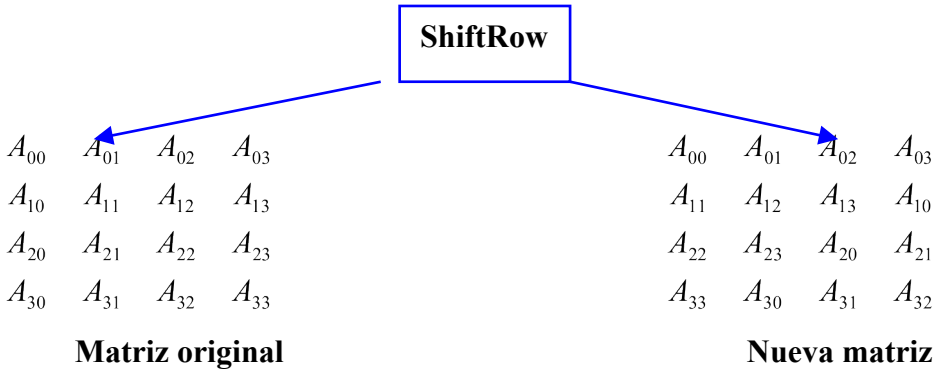

Figura 2. Transformación ShiftRow

3.3.Transformación MixColumns

Se trata de la transformación siguiente a ShiftRow. MixColumns opera columna por columna de la matriz, tomando a cada una como un polinomio de grado tres. Es decir, que las columnas son consideradas como polinomios en el campo $GF(2^8)$ y cada una es multiplicada por una fila de la siguiente matriz Ref(x):

$$\text{Ref(x)}=\begin{matrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{matrix} \qquad \text{Ec. (2)}$$

O sea, que la multiplicación de la primer columna de la matriz A con la primer fila de la matriz Ref(x), da por resultado al elemento de subíndice 00 de la nueva matriz. De la misma manera, la multiplicación de la primer columna de la matriz A con la segunda fila de la matriz Ref(x) da por resultado al elemento de subíndice 10 de la nueva matriz y así sucesivamente. En caso de que la multiplicación de un elemento de la matriz A con un elemento de la matriz Ref(x), de por resultado un polinomio de grado ocho, entonces a dicho polinomio hay que dividirlo por $x^8 + x^4 + x^3 + x + 1$, y tomar el resto de esta división como resultado.

Por ejemplo, si la matriz A luego de aplicar ShiftRow es:

$$A=\begin{matrix} 52 & 36 & 85 & bc \\ 33 & a5 & 50 & 9f \\ a8 & d0 & 9f & f9 \\ 76 & aa & 02 & bc \end{matrix}$$

Para obtener el elemento de subíndice 10 de la nueva matriz, habrá que multiplicar la primer columna de la matriz A con la segunda fila de la matriz Ref(x), obteniéndose el siguiente resultado:

(52*01) xor (33*02) xor (a8*03) xor (76*01) Ec. (3)

Estos números pueden verse como polinomios:

52=01010010 => $x^6 + x^4 + x$ Ec. (4)

01=00000001=> 1 Ec. (5)

Así, la multiplicación de $x^6 + x^4 + x$ por 1, es igual al mismo polinomio, con lo cual el resultado en binario es **01010010**.

33=00110011=> $x^5 + x^4 + x + 1$ Ec. (6)

02=00000010=> x Ec. (7)

Luego, la multiplicación de ambos polinomios es igual a: $x^6 + x^5 + x^2 + x$, que, expresado en formato binario es **01100110**.

a8=10101000=> $x^7 + x^5 + x^3$ Ec. (8)

03=00000011=> $x + 1$ Ec. (9)

De esta manera, la multiplicación de los dos polinomios da como resultado $x^8 + x^7 + x^6 + x^5 + x^4 + x^3$. Como en este caso el polinomio resultante es de grado ocho, hay que dividirlo por $x^8 + x^4 + x^3 + x + 1$. Para ello hay que completar al dividendo con ceros,

con lo cual, el cociente es igual a uno y el resto es igual a $x^7 + x^6 + x^5 + x + 1$, cuyo formato en binario es **11100011**.

76=01110110=> $x^6 + x^5 + x^4 + x^2 + x$ Ec. (10)

01=00000001=> 1 Ec. (11)

La multiplicación de $x^6 + x^4 + x$ por 1, es igual al mismo polinomio, con lo cual el resultado en binario es **01110110**.

Por último, realizando la or-exclusiva de los cuatro resultados: **(01010010)** xor **(01100110)** xor **(11100011)** xor **(01110110)=10100001=a1**. El resultado obtenido sería el elemento A_{10} de la nueva matriz transformada por MixColumns. Esta operación se repite con los demás elementos de la matriz.

3.4.Expansión de la clave

La expansión de la clave consiste en la generación de diez claves distintas, ya que el proceso de encriptado mencionado previamente se repetirá diez veces o, como se describe en el algoritmo, a lo largo de diez round. De esta manera se genera una clave diferente para cada round.

En el proceso de expansión, se utilizan las transformaciones: **RotWord y SubWord.** La primera consiste en tomar una palabra de cuatro bytes como entrada y realizar una permutación cíclica sobre la misma. El resultado de esta rotación se suma en módulo-2 (EXOR) con una constante Rcon[i], que contiene los valores dados por [x^{i-1}, {00}, {00}, {00}], siendo x^{i-1} potencias de x (ó {02} en GF(2^8)). SubWord es una función que toma 4 bytes de entrada y aplica sobre los mismos la S-box.

Ejemplo de expansión:

Sea la siguiente clave: 2b7e151628aed2a6abf7158809cf4f3c, sobre la que se aplicará el proceso de expansión para obtener las 10 subclaves necesarias.

Primer se divide la clave en palabras de cuatro bytes ($N_k = 4$) según se indica en las siguientes ecuaciones:

w0=2b7e1516 Ec. (11)

w1=28aed2a6 Ec. (12)

w2=abf71588 Ec. (13)

w3=09cf4f3c Ec. (14)

El siguiente esquema representa el procesamiento completo de la clave original:

i	Palabra de 4 bytes	Después de RotWord	Después de SubWord	Num$[i/N_k]$	Después de XOR con Num	W$[i - N_k]$	W[i]=Num XOR w[$i - N_k$]
4	09cf4f3c	cf4f3c09	8a84eb01	01000000	8b84eb01	2b7e1516	a0fafe17
5	a0fafe17					28aed2a6	88542cb1
6	88542cb1					abf71588	23a33939
7	23a33939					09cf4f3c	2a6c7605
8	2a6c7605	6c76052a	50386be5	02000000	52386be5	a0fafe17	f2c295f2
9	f2c295f2					88542cb1	7a96b943
10	7a96b943					23a33939	5935807a
11	5935807a					2a6c7605	7359f67f
12	7359f67f	59f67f73	cb42d28f	04000000	cf42d28f	f2c295f2	3d80477d
13	3d80477d					7a96b943	4716fe3c
14	4716fe3c					5935807a	1e237e44
15	1e237e44					7359f67f	6d7a883b
16	6d7a883b	7a883b6d	dac4e23c	08000000	d2c4e23c	3d80477d	Ef44a541
17	ef44a541					4716fe3e	a8525b7f
18	a8525b7f					1e237e44	b671253b
19	b671253b					6d7a883b	Db0bad00
20	db0bad00	0bad00db	**2b9563b9**	10000000	3b9563b9	ef44a541	d4d1c6f8
21	d4d1c6f8					a8525b7f	7c839d87
22	7c839d87					b671253b	caf2b8bc
23	caf2b8bc					db0bad00	11f915bc
i	**Palabra de 4 bytes**	**Después de RotWord**	**Después de SubWord**	**Num**$[i/N_k]$	**Después de XOR con Num**	**W**$[i - N_k]$	**W[i]=Num XOR w[**$i - N_k$]
24	11f915bc	F915bc11	99596582	20000000	b9596582	d4d1c6f8	6d88a37a

25	6d88a37a					7c839d87	110b3efd
26	110b3efd					caf2b8bc	dbf98641
27	dbf98641					11f915bc	Ca0093fd
28	ca0093fd	0093fdca	63dc5474	40000000	23dc5474	6d88a37a	4e54f70e
29	4e54f70e					110b3efd	5f5fc9f3
30	5f5fc9f3					dbf98641	84a64fb2
31	84a64fb2					ca0093fd	4ea6dc4f
32	4ea6dc4f	a6dc4f4e	2486842f	80000000	a486842f	4e54f70e	ead27321
33	ead27321					5f5fc9f3	b58dbad2
34	b58dbad2					84a64fb2	312bf560
35	312bf560					4ea6dc4f	7f8d292f
36	7f8d292f	8d292f7f	5da515d2	1b000000	46a515d2	ead27321	Ac7766f3
37	ac7766f3					b58dbad2	19fadc21
38	19fadc21					312bf560	28d12941
39	28d12941					7f8d292f	575c006e
40	575c006e	5c006e57	4ª639f5b	36000000	7c639f5b	ac7766f3	d014f9a8
41	d014f9a8					19fadc21	c9ee2589
42	c9ee2589					28d12941	e13f0cc8
43	e13f0cc8					575c006e	b6630ca6

Tabla 2. Ejemplo de Expansión de la clave

Las subclaves obtenidas se escriben en la última columna. Por ejemplo la primera que se obtiene es la a0fafe17 88542cb1 23a33939 2a6c7605, la segunda f2c295f2 7a96b943 5935807a 7359f67f, y así sucesivamente.

Como se observó en la **Fig. 1**, la primera operación que se realiza es AddRoundKey, es decir, se realiza una EXOR entre el texto plano y la clave a utilizar. Por ejemplo si los 16 caracteres del texto plano son: 00112233 44556677 8899aabb ccddeeff, y la clave es de la forma: 00010203 040506070 8090a0b0 c0d0e0f, entonces una EXOR entre estos dos valores da como resultado: 00102030 40506070 8090a0b0 c0d0f0. Este resultado puede expresarse en forma matricial:

$$\begin{matrix} 00 & 40 & 80 & c0 \\ 10 & 50 & 90 & d0 \\ 20 & 60 & a0 & e0 \\ 30 & 70 & b0 & f0 \end{matrix} \qquad \text{Ec. (15)}$$

Luego a dicha expresión se la aplica la transformación **ByteSub**, con lo cual se obtiene: 63cab704 0953d051 cd60e0e7 ba70e18c, donde al número 63, se obtuvo de ingresar a la tabla S-box, con el subíndice 00, y de igual manera se procedió con los demás números. La siguiente transformación a aplicar es **ShiftRow**, que consiste en desplazar a las tres últimas filas, dando como resultado 6353e08c 0960e104 cd70b751 bacad0e7. Después se aplica la transformación **MixColumns**, obteniéndose: 5f726415 57f5bc92 f7be3b29 1db9f91a. Restaría efectuar la EXOR con la primera clave generada en la **Expansión de la clave** y a este resultado se le vuelve aplicar las tres transformaciones anteriores. Luego se vuelve a realizar una EXOR con la segunda clave generada en la **Expansión de la clave** y se repite lo mismo durante nueve ciclos. En el décimo ciclo se presenta una diferencia pues la única operación que no se realiza es la **MixColumns**, con lo cual la EXOR es realizada entre el resultado de la aplicación de **ShiftRow** y la última clave generada en la **Expansión de la clave**.

3.5.Proceso de desencriptado

Ahora lo que se va a analizar, es el proceso inverso al encriptado, es decir, el desencriptado. A dicho proceso se lo puede estudiar con el siguiente diagrama de flujo:

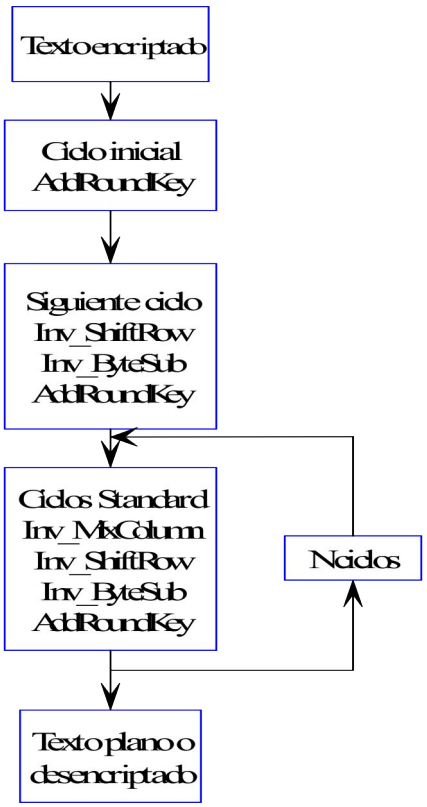

Figura 3. Proceso de desencriptado

La transformación **AddRoundKey**, igual que en el proceso de encriptado, consiste de una EXOR de 128 bits. A continuación se analizarán las otras transformaciones, inversas de aquellas aplicadas en el proceso de encriptado.

3.5.1.Transformación Inv-ShiftRow

Esta transformación es la inversa de la transformación **ShiftRow**. En la misma, los bytes de las tres últimas filas son desplazados cíclicamente en distintos números de posiciones (offset). En la primer fila el offset es nulo, es decir, que la fila no es desplazada. En la segunda fila, el desplazamiento es de 3 bytes, en la tercera es de 2 bytes y en la última es de 1byte, en todos los casos hacia la izquierda. Este procesamiento se ilustra gráficamente en la **Fig. 4.**

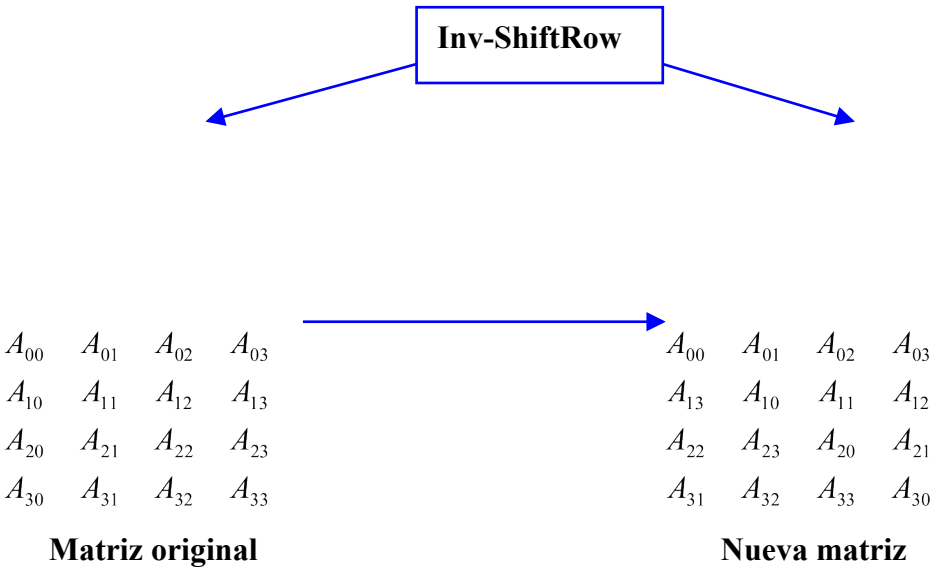

Figura 4. Transformación Inv-ShiftRow

3.5.2.Transformación Inv-ByteSub

Esta transformación es la inversa de la transformación **ByteSub**. La misma es una sustitución no lineal, que opera independientemente en cada byte de la matriz, y se puede realizar usando la matriz inversa de S-box.

La Tabla 2 muestra la inversa de la matriz S-box en formato hexadecimal.

		y															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**	**f**
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8 1ce8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b

	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Tabla 3. **Inv-ByteSub**

Inv-ByteSub toma un elemento en formato hexadecimal de la matriz A de cuatro filas por cuatro columnas, y dicho número pasa a formar parte del subíndice de la inversa de la matriz **S-box**. Por ejemplo, si el número de la matriz A es {12} en formato hexadecimal, el mismo pasa a ser reemplazado por {39} en el mismo formato, ingresando primero por la columna **x** y luego por la columna **y**, es decir, que el número a ser reemplazado ingresa a la matriz **S-box** como subíndice.

3.5.3.Transformación Inv-MixColumn

Esta transformación es la inversa de la transformación **MixColumn**. La misma también opera columna por columna de la matriz, tomando a cada una como un polinomio de grado tres. Es decir, que las columnas son consideradas como polinomios en el campo $GF(2^8)$ y cada una es multiplicada por una fila de la siguiente matriz Inv_Ref(x):

$$\text{Inv\_Ref(x)}=\begin{matrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{matrix} \qquad \text{Ec. (16)}$$

Igual que en la transformación MixColumn, la multiplicación de la primer columna de la matriz A con la primer fila de la matriz Inv_Ref(x), da por resultado al elemento de subíndice 00 de la nueva matriz. La multiplicación de la primer columna de la matriz A con la segunda fila de la matriz Inv_Ref(x) da por resultado al elemento de subíndice 10 de la nueva matriz y así sucesivamente. En caso de que la multiplicación de un elemento de la matriz A con un elemento de la matriz Inv_Ref(x), de por resultado un polinomio de grado ocho, entonces a dicho polinomio hay que dividirlo por $x^8 + x^4 + x^3 + x + 1$, (el mismo polinomio utilizado en MixColumns), y lo que se toma como resultado es el resto de dicha división.

Por ejemplo, si la matriz A luego de aplicar AddRoundKey es:

$$\begin{matrix} 43 & cd & 48 & e2 \\ 90 & 65 & 0e & 84 \\ 5b & 09 & fb & 74 \\ b3 & 7d & 9d & 1a \end{matrix} \qquad \text{Ec. (17)}$$

Si se desea obtener al elemento de subíndice 00 de la nueva matriz, al multiplicar la primer columna de la matriz A con la primera fila de la matriz Inv_Ref(x), se obtiene lo siguiente:

(43*0e) xor (90*0b) xor (5b*0d) xor (b3*09) Ec. (18)

De donde:

43=01000011 => $x^6 + x + 1$ Ec. (19)

0e=00001110=> $x^3 + x^2 + x$ Ec. (20)

Luego, la multiplicación de los dos polinomios es $x^9 + x^8 + x^7 + x^4 + x$, como el polinomio resultante es de grado nueve, entonces hay que dividirlo por $x^8 + x^4 + x^3 + x + 1$. Para realizar dicha división hay que completar al dividendo con ceros (igual que en **MixColumn**), con lo cual el cociente es igual a x+1 y el resto es $x^7 + x^5 + x^4 + x^3 + x^2 + x + 1$, cuyo formato en binario es **10111111**.

90=10010000 => $x^7 + x^4$ Ec. (20)

0b=00001011=> $x^3 + x + 1$ Ec. (21)

La multiplicación de ambos polinomios es $x^{10} + x^8 + x^5 + x^4$, como el polinomio resultante es de grado décimo, entonces hay que volver a dividirlo por el polinomio $x^8 + x^4 + x^3 + x + 1$. El cociente es igual a $x^2 + 1$ y el resto es $x^6 + x^2 + x + 1$, cuyo formato en binario es **01000111**.

5b=01011011 => $x^6 + x^4 + x^3 + x + 1$ Ec. (22)

0d=00001101=> $x^3 + x^2 + 1$ Ec. (23)

La multiplicación da $x^9 + x^8 + x^7 + x^6 + x^5 + x^3 + x^2 + x + 1$, como el polinomio resultante es de grado nueve, entonces hay que volver a dividirlo por el polinomio $x^8 + x^4 + x^3 + x + 1$. El cociente es igual a x+1 y el resto es $x^7 + x^6 + x$, cuyo formato en binario es **11000010**.

b3=10110011 => $x^7 + x^5 + x^4 + x + 1$ Ec. (24)

09=00001001=> $x^3 + 1$ Ec. (25)

La multiplicación da $x^{10} + x^8 + x^5 + x^3 + x + 1$ y, como en los casos anteriores, hay que volver a dividirlo por el polinomio $x^8 + x^4 + x^3 + x + 1$. El cociente resulta ser $x^2 + 1$ y el resto $x^6 + x^4 + x^3 + x^2$, cuyo formato en binario es igual a **01011100**.

Realizando la EXOR de los cuatro resultados: **(10111111)** xor **(01000111)** xor **(11000010)** xor **(01011100)=01100110=66.** Este resultado se convierte en el elemento A_{00} de la nueva matriz transformada por Inv-MixColumn. Las mismas operaciones se realizan con los otros elementos de la matriz.

Como se observa en el diagrama de la operación de desencriptado, (Fig. 4), la primera operación que se realiza es AddRoundKey, es decir, la EXOR entre el texto encriptado y la última clave obtenida en la **Expansión de la clave**. Por ejemplo si los 16 caracteres del texto encriptado son los siguientes: 69c4e0d8 6a7b0430 d8cdb780 70b4c55a, y la última clave obtenida en la **Expansión de la clave** es: 13111d7f e3944a17 f307a78b 4d2b30c5, entonces una XOR entre estos dos valores da como resultado: 7ad5fda7 89ef4e27 2bca100b 3d9ff59f, que expresada en forma matricial sería:

$$\begin{matrix} 7a & 89 & 2b & 3d \\ d5 & ef & ca & 9f \\ fd & 4e & 10 & f5 \\ a7 & 27 & 0b & 9f \end{matrix} \qquad \text{Ec. (26)}$$

Luego, a dicho resultado, sólo en el primer ciclo se la aplica la transformación **Inv_ShiftRow**, la cual consiste en desplazar a las tres últimas filas dando como resultado 7a9f1027 89d5f50b 2beffd9f 3dca4ea7. La siguiente transformación es **Inv_ByteSub**, con lo cual se obtiene: bd6e7c3df2 b5779e0b 61216e8b 10b689, donde al número bd, se obtuvo de ingresar a la tabla de la inversa de la matriz S-box, con el subíndice 00, y de la misma forma se trabaja con los demás números. A continuación a dicha expresión se le realiza una EXOR con la anteúltima clave generada en la **Expansión de la clave,** 549932d1 f0855768 1093ed9c be2c974e y a la nueva expresión que resulta de dicha operación se le aplica la transformación **Inv_MixColumns**, obteniéndose como resultado: 54d990a1 6ba09ab5 96bbf40e a111702f, después a dicha expresión se le aplica la transformación **Inv_ShiftRow**, después la transformación **Inv_ByteSub**, y por último se realiza una EXOR entre la última expresión obtenida y la octava clave obtenida en **la Expansión de la clave**, y se vuelven a aplicar las transformaciones en este orden: **Inv_MixColumns**, **Inv_ShiftRow**, **Inv_ByteSub** y **AddRoundKey** o XOR entre la expresión obtenida después de **Inv_ByteSub** y la séptima clave obtenida en la **Expansión de la clave**, luego se repiten los mismos pasos, hasta completar los diez ciclos. Se observa que en este proceso de desencriptado, las claves

obtenidas en la **Expansión de la clave**, son utilizadas en orden decreciente, es decir, desde la última hasta la primera obtenidas en el proceso de encriptado. También es de observar que la única diferencia en el orden de las transformaciones, se presenta en el primer ciclo, donde no se aplica la transformación **Inv_MixColumns**, sino sólo las transformaciones **Inv_ShiftRow** e **Inv_ByteSub**, luego de realizar la EXOR entre el texto encriptado y la última clave generada en la **Expansión de la clave**.

SECCIÓN IV

DESCRIPCIÓN DEL SOFTWARE

4.Descripción del Software

El presente software fue realizado en C++Builder, por ser el lenguaje C uno de los lenguajes más potentes y con mayor capacidad en el manejo de memoria y de interrupciones.

El software puede considerarse dividido en dos partes principales:

Un archivo de extensión .h, que contiene todas las funciones de encriptado y desencriptado a utilizar, además de las matrices necesarias para realizar ciertas transformaciones, como es el caso de la **S-box** y su inversa para las transformaciones **ByteSub** y su inversa respectivamente. Los llamados de las funciones de encriptado y desencriptado se encuentran agrupados a su vez, en dos funciones denominadas **EncriptarVector** y **DesencriptarVector**, que realizan los procesos de encriptado y desencriptado, explicados en las secciones 3 y 3.5, respectivamente.

El otro archivo, es un archivo de extensión **.cpp**, que contiene en su cabecera el llamado del archivo de extensión **.h**, mencionado anteriormente, y la programación de la interfaz gráfica, donde en la programación del botón **Encriptar archivo** y del botón **Desencriptar archivo**, se encuentran los llamados a las funciones **EncriptarVector** y **DesencriptarVector**, respectivamente. La interfaz gráfica puede dividirse en dos partes, una correspondiente al panel de encriptado y la otra al panel de desencriptado, en los cuales se encuentran los botones **Encriptar archivo** y **Desencriptar archivo**, respectivamente. Los dos paneles han sido diseñados con el objetivo de ofrecer una interfaz de interpretación sencilla y uso simple. Estas características se asocian a la flexibilidad que ofrece el programa en cuanto a que, una vez ingresada la clave, el usuario pueda no sólo elegir el archivo a encriptar o desencriptar sino también su ubicación en un directorio destino. En la cabecera del archivo encriptado, también figuran encriptados el tamaño del archivo a encriptar, el tamaño y nombre de la extensión, para que el usuario pueda decodificarlos y desencriptar al archivo con la misma extensión original.

Los pasos a seguir en el intercambio entre usuario e interfaz gráfica, se presentan en la **Fig. 5**, ya sea en el proceso de encriptado como en el de desencriptado.

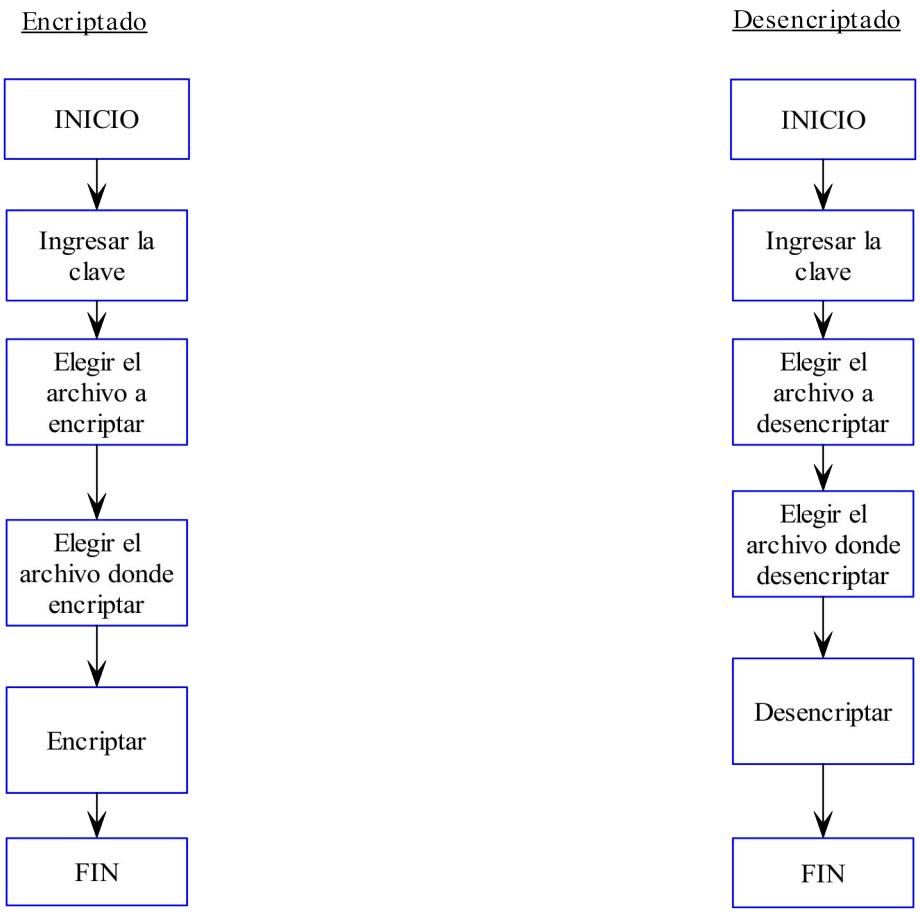

Figura 5.Interfaz gráfica para el proceso de encriptado y desencriptado

Las funciones **EncriptarVector** y **DesencriptarVector**, poseen como parámetro el vector que contiene los 16 caracteres a encriptar o desencriptar, la matriz correspondiente a la clave y el vector **vectordeclave**, en el cual se almacena la clave cuando el usuario la ingresa en el formulario en la caja de texto correspondiente. Es decir, una vez que el usuario ingresa la clave, el software la almacena en la matriz que se usa como parámetro, y que también será usada como parámetro en las demás funciones a utilizar.

El esquema de la **Fig. 6** pretende resaltar las estructuras de la función **EncriptarVector**:

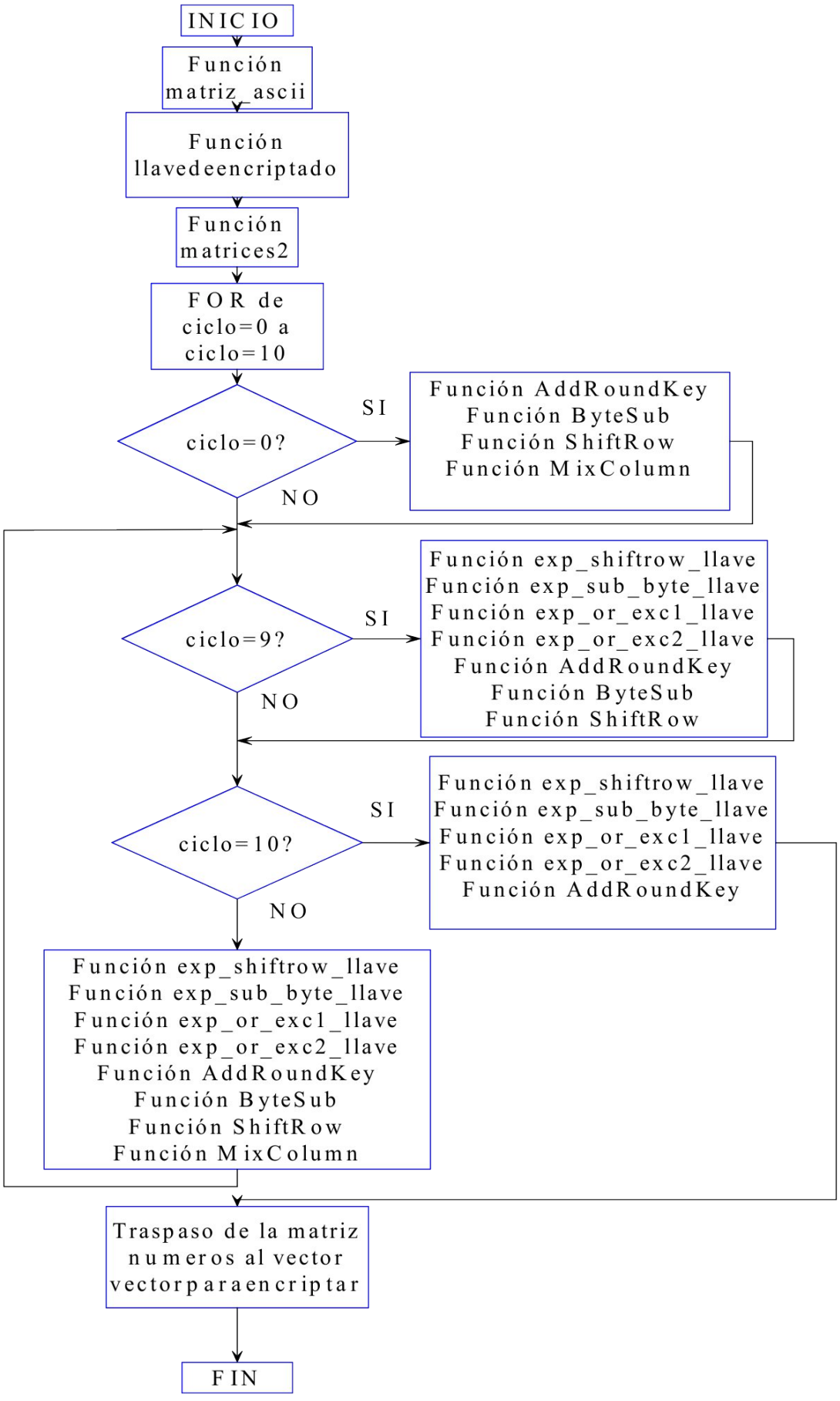

Figura 6.Diagrama de flujo correspondiente a la función EncriptarVector

La función **matriz_ascii** traspasa los 16 bytes que se encuentran almacenados en el vector **vectorparaencriptar** de 16 componentes, a la matriz **números** de cuatro filas por cuatro columnas. Es decir, que el primer paso del programa consiste en abrir al archivo a encriptar, tomar de este los datos en grupos de 16 bytes o caracteres y luego almacenarlos en dicho vector, para posteriormente ser utilizado como parámetro de la función **EncriptarVector.** En definitiva los datos a encriptar ingresan a la función como parámetros del vector **vectorparaencriptar**, para luego ser asignados a la matriz **números**, la cual es utilizada como parámetro en el resto de las funciones de encriptado. Al finalizar la rutina de la función **EncriptarVector**, los valores vuelven a ser asignados al vector **vectorparaencriptar**, para posteriormente almacenarse en el archivo donde se guardará el texto encriptado. El siguiente diagrama de flujo es un resumen de lo expuesto en este ítem:

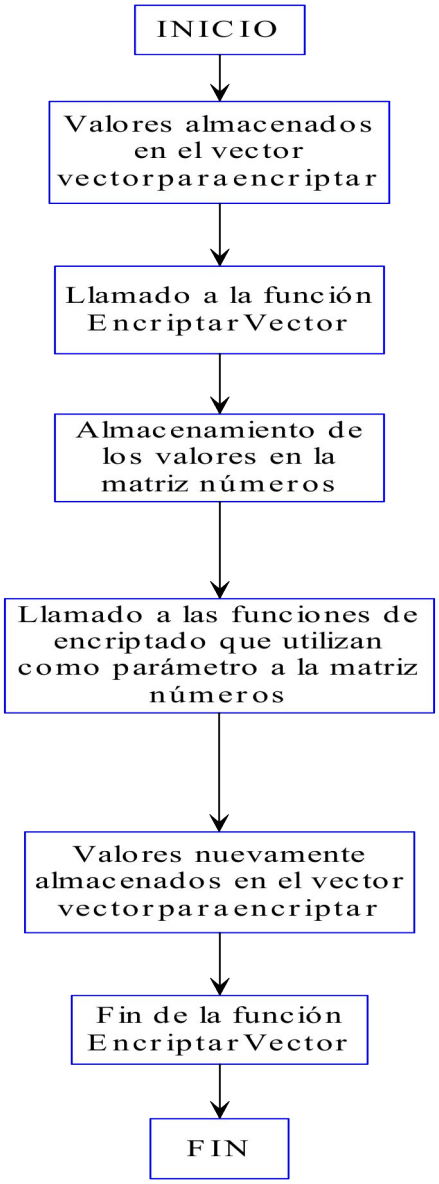

Figura 7.Explicación del uso del vector vectorparaencriptar como parámetro de la función EncriptarVector.

La función **llavedeencriptado** por su parte, traspasa la clave almacenada en el vector **vectordeclave**, a la matriz **llave** de cuatro filas por cuatro columnas. La misma será utilizada como parámetro en las funciones de encriptado. Cuando el usuario escribe la clave en la caja de texto el software la almacena en dicho vector.

La función **matrices2** declara los valores de la matriz **S-box**, e inicia al vector Num[i / N_k], con el valor 01000000. Este vector es luego utilizado en la **Expansión de la clave**.

La función **exp_shiftrow_llave** es la transformación **ShiftRow**, pero aplicada a la clave en el proceso de **Expansión de la clave**. Se desplaza las filas cíclicamente y también se la denomina **RotWord**.

Otra transformación que se utiliza en la **Expansión de la clave** es **ByteSub** o **SubWord**, que en este caso se denomina **exp_sub_byte_llave**.

La función **exp_or_exc1_llave** genera al vector **Num[**i / N_k], en sus distintos valores como se observa en la **Tabla 2** y luego realiza una EXOR entre cada uno de dichos valores y el vector que se obtuvo luego de la transformación **ByteSub**, aplicada a la clave.

Por último la función **exp_or_exc2_llave** realiza una EXOR entre el valor obtenido anteriormente y el vector W[$i - N_k$], como se observa en la **Tabla 2**.

Luego del décimo round, los valores de la matriz **números** vuelven a ser asignados al vector **vectorparaencriptar**, el cual se almacena a su vez en el archivo en el cual se guardará al texto encriptado.

En la cabecera del archivo encriptado, figuran los siguientes datos encriptados en ese orden:

El tamaño del archivo.

El tamaño de la extensión del archivo.

La extensión del archivo.

Cada uno de los tres valores mencionados, se almacenan en el vector **vectorparaencriptar**, para ser pasado como parámetro de la función **EncriptarVector.** Cada uno de estos tres valores se extraen del archivo a encriptar durante la programación del botón **Encriptar archivo** del archivo de extensión **. cpp**. En dicha función se invoca a la función **EncriptarVector**, cada vez que se encripta a cada uno de estos tres valores. En el archivo que va a contener al texto encriptado, en su cabecera, se reservarán:

16 bytes para el tamaño del archivo.

16 bytes para el tamaño de la extensión del archivo.

16 bytes para la extensión del archivo.

La estructura de la programación del botón **Encriptar archivo**, se representa en la **Fig. 8**.

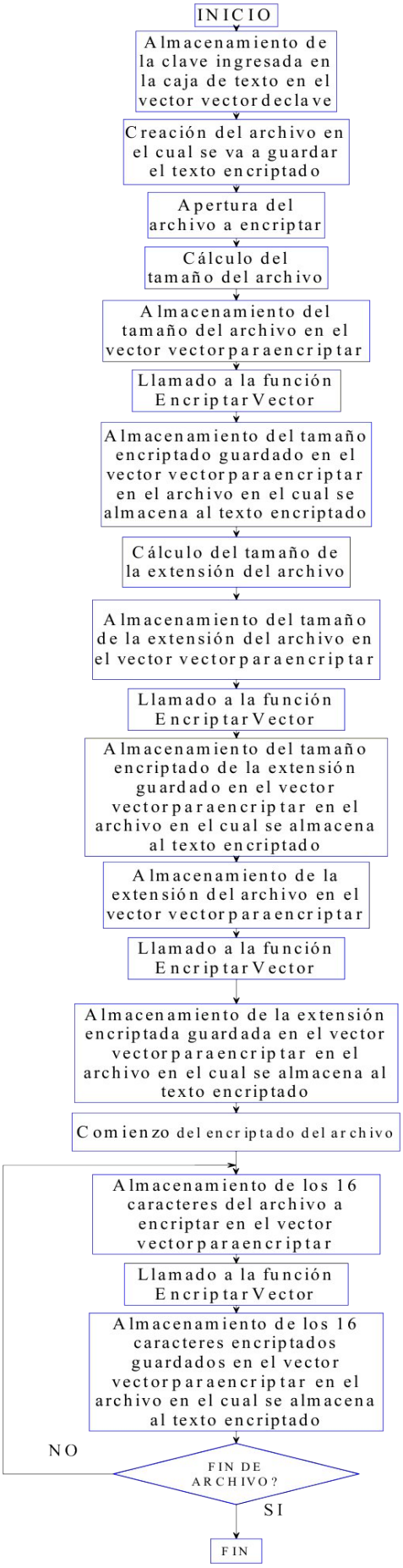

Figura 8.Estructura de programación del botón Encriptar archivo

En el proceso de desencriptado, se utiliza la función **DesencriptarVector**, que utiliza como parámetros el vector **HexParaDesencriptar**, en el que se almacenan los 16 bytes del texto encriptado, la matriz de almacenamiento de la clave y al vector **vectordeclave,** depósito inicial de la clave cuando el usuario la ingresa en el formulario en la caja de texto.

La función **DesencriptarVector,** puede interpretarse fácilmente mediante el esquema de la **Fig. 9**.

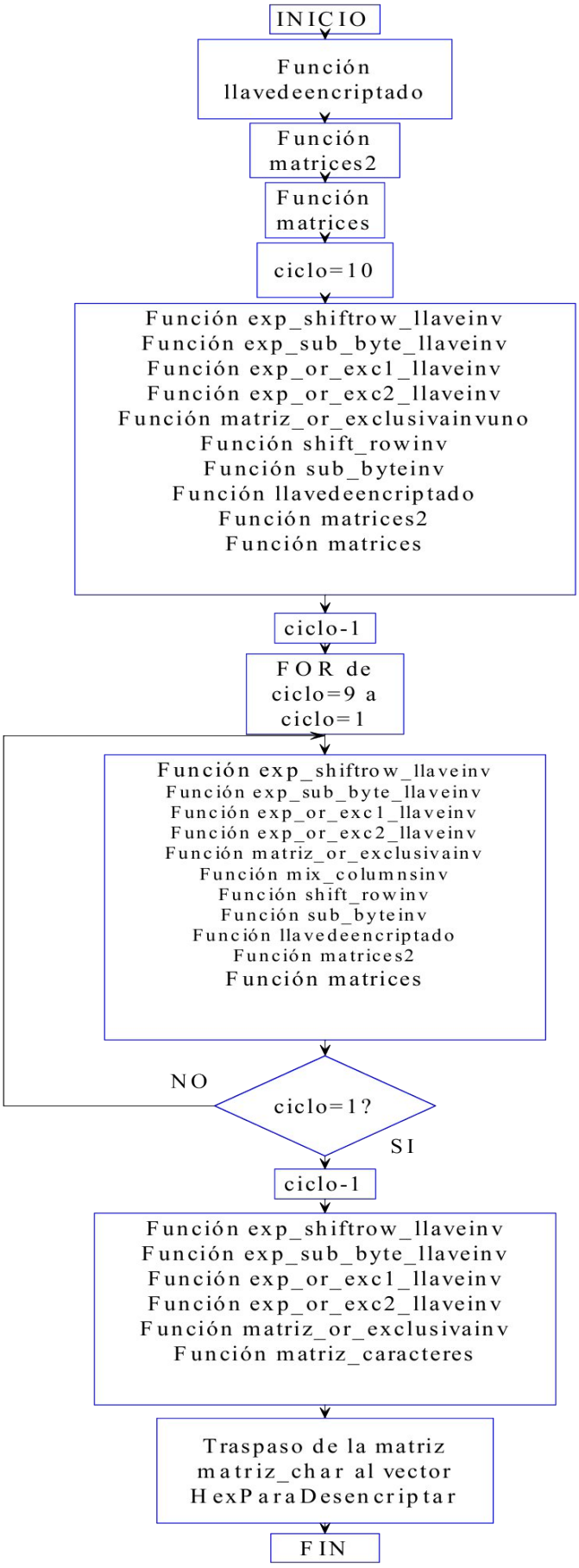

Figura 9.Diagrama de flujo correspondiente a función DesencriptarVector

De la misma manera que en el caso de la función de encriptado, la función l**lavedeencriptado** es la encargada de traspasar la clave almacenada en el vector **vectordeclave** a la matriz **llave** de cuatro filas por cuatro columnas que luego será utilizada como parámetro en las funciones de desencriptado. Cuando el usuario escribe la clave en la caja de texto el software la almacena en dicho vector.

La función **matrices2** declara los valores de la matriz **S-box**, e inicializa al vector **Num**[i / N_k] con el valor 01000000, que se utilizará en la **Expansión de la clave**.

La función **matrices** declara los valores de la inversa de la matriz **S-box**.

La función **exp_shiftrow_llaveinv**, es igual a la función **exp_shiftrow_llave**, la cual es la transformación **ShiftRow**, pero aplicada a la clave en el proceso de **Expansión de la clave**. Recordar que en el proceso de desencriptado, la primer clave a utilizar en la **Expansión de la clave**, es la última clave generada en el proceso de encriptado.

La función **exp_sub_byte_llaveinv** es igual a la función **exp_sub_byte_llave**, la cual es la transformación **ByteSub**, aplicada a la clave.

La función **exp_or_exc1_llaveinv**, es igual a la función **exp_or_exc1_llave**, que genera el vector **Num**[i / N_k], en sus distintos valores como se observa en la **Tabla 2,** y luego realiza la EXOR entre dicho vector y el vector que se obtuvo luego de la transformación **ByteSub**, aplicada a la clave.

La función **exp_or_exc2_llaveinv** es igual a la función **exp_or_exc2_llave:** realiza una EXOR entre el valor obtenido anteriormente y el vector W[$i - N_k$], como se observa en la **Tabla 2**.

La función **matriz_or_exclusivainvuno** realiza una EXOR entre el vector **HexParaDesencriptar**, usado como parámetro en la función **DesencriptarVector**, y la clave obtenida luego de la transformación anterior. El resultado es almacenado en la matriz **números**, la cual es utilizada como parámetro en las funciones que siguen en el proceso de desencriptado: **shift_rowinv**, **sub_byteinv** y **mix_columnsinv**, explicadas anteriormente.

La función **matriz_or_exclusivainv**, también realiza una EXOR entre la matriz **números** (que tiene almacenado a los 16 caracteres del texto encriptado), y la matriz que tiene almacenada a la clave en sus distintas etapas de expansión.

La función **matriz_caracteres**, almacena a la matriz **números** desencriptada, es decir, la que se obtiene en la última fase del desencriptado en la matriz **matriz_char** de

cuatro filas por cuatro columnas, de tipo **unsigned char.** Así se obtienen los caracteres correspondientes al texto desencriptado.

Luego del décimo round, los valores de la matriz **matriz_char** vuelven a ser asignados al vector **HexParaDesencriptar**, utilizado como parámetro de la función **DesencriptarVector**. De esta forma los valores ingresan a la función a través del vector **HexParaDesencriptar** para luego ser asignados a una matriz, que luego se utiliza como parámetro en todas las demás funciones de desencriptado. Al finalizar la rutina de la función **DesencriptarVector**, los valores vuelven a ser asignados al vector **HexParaDesencriptar**, que se almacenará en el archivo de texto desencriptado. El esquema de la **Fig. 10** representa un resumen de este proceso.

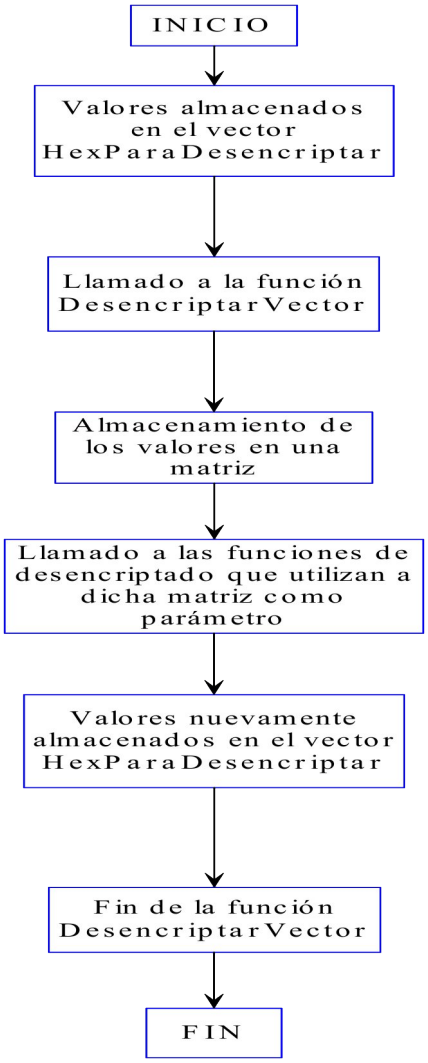

Figura 10. Explicación del uso del vector HexParaDesencriptar como parámetro de la función DesencriptarVector.

Como se explicó antes, en la cabecera del archivo encriptado, se encuentran encriptados, el **tamaño del archivo**, el **tamaño de la extensión del archivo** y **la extensión del archivo.** Estos valores serán desencriptados en el mismo orden. Cada uno se extrae del

archivo a desencriptar durante la programación del botón **Desencriptar archivo**, perteneciente al archivo de extensión **. cpp.** La misma invoca a la función **DesencriptarVector**, cada vez que se desencripta a cada uno de estos tres valores.

La estructura de la programación del botón **Desencriptar archivo** se presenta en la **Fig. 11**.

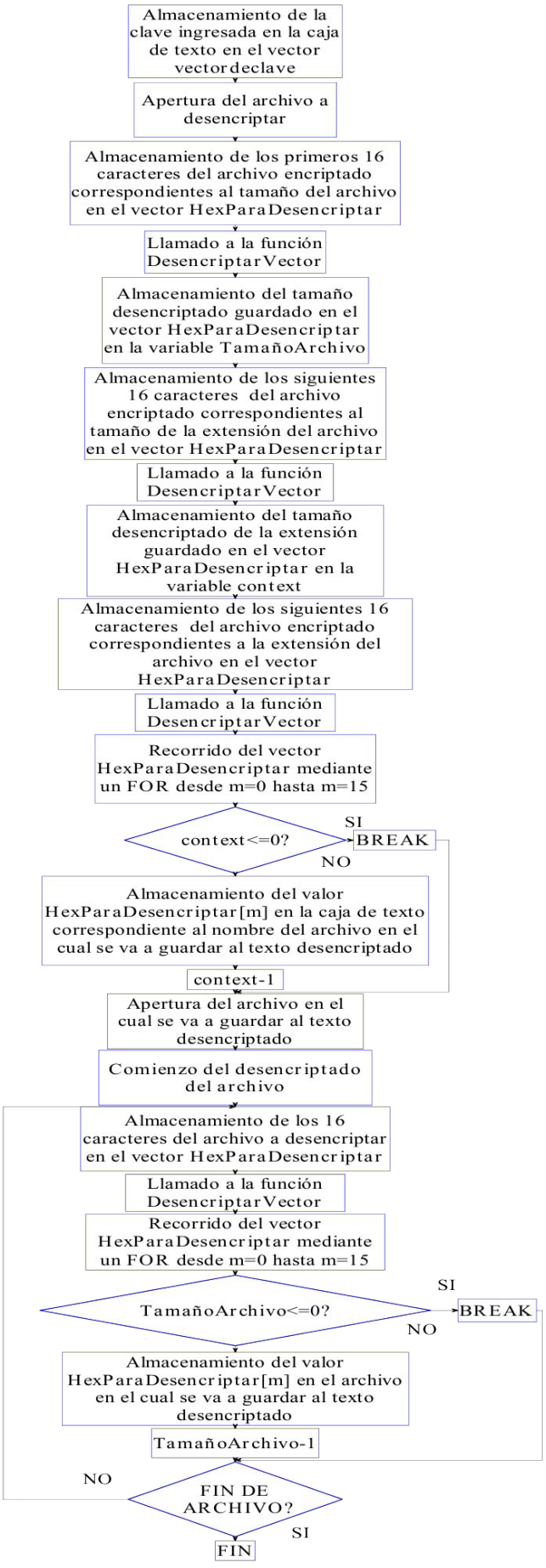

Figura 11.Estructura del botón Desencriptar archivo

Como se observa en el diagrama de flujo, luego de desencriptar el tamaño del archivo y el tamaño de la extensión, se procede a desencriptar la extensión del archivo. Esta última se guardará en la caja de texto en la que el usuario ingresa el nombre del archivo para almacenar el texto a desencriptar. Es decir, una vez que el nombre del archivo destino es ingresado y se presiona el botón **Desencriptar archivo**, la extensión del archivo a encriptar o archivo origen es concatenada con el nombre del archivo destino. De esa forma el archivo en cuestión conserva la extensión original y puede, luego del proceso de desencriptado, ser interpretado correctamente por la aplicación correspondiente. Es de resaltar que, a medida que un carácter de la extensión es almacenado en la caja de texto, el contador **context** (que guarda al tamaño de la extensión desencriptado previamente), es decrementado en una unidad, para seguir el control de la concatenación.

En el desencriptado del archivo, cada vez que un carácter del vector **HexParaDesencriptar** (el cual contiene a los caracteres ya desencriptados), es almacenado en el archivo destino, el contador **TamañoArchivo** (el cual contiene el tamaño del archivo original), es decrementado en una unidad, para que de esa manera no sean guardados en el archivo destino caracteres adicionales no existentes en el archivo a desencriptar.

SECCIÓN V

MANUAL DE USO

5.Manual de uso

5.1.Sobre el presente manual

5.1.1.Objetivo

El objetivo del presente manual es presentar al usuario la interfaz gráfica del programa, así como su forma de uso.

5.1.2.Restricciones

En cuanto a la instalación, se puede mencionar que el software puede ser instalado en Windows 95, 98, 2000 Pro y Windows XP.

5.2.Interfaz gráfica

A la misma se la puede dividir en tres partes:

Menú de presentación.

Panel de encriptado.

Panel de desencriptado.

El primer item con el cual el usuario se encuentra, es el Menú de presentación, en el cual presionando el botón **Siguiente**, aparece el panel de encriptado y en la parte superior del formulario, un menú de opciones, donde el usuario puede elegir que panel desea utilizar.

A continuación se muestra al menú de presentación:

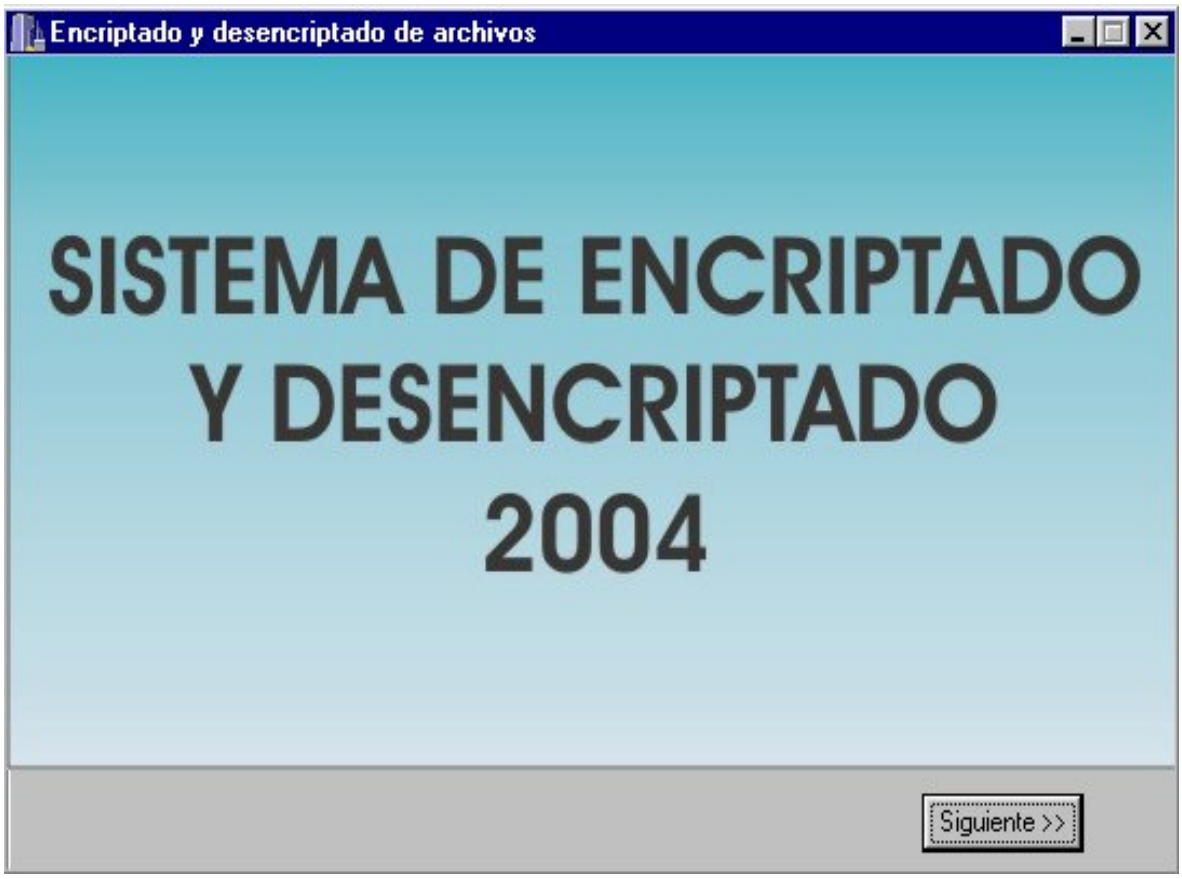

Imagen.1.Menú de presentación

5.2.1.Panel de encriptado

Presionando el botón **Siguiente** o haciendo clic con el mouse en **Encriptar**, aparece el panel de encriptado, el cual se lo muestra a continuación:

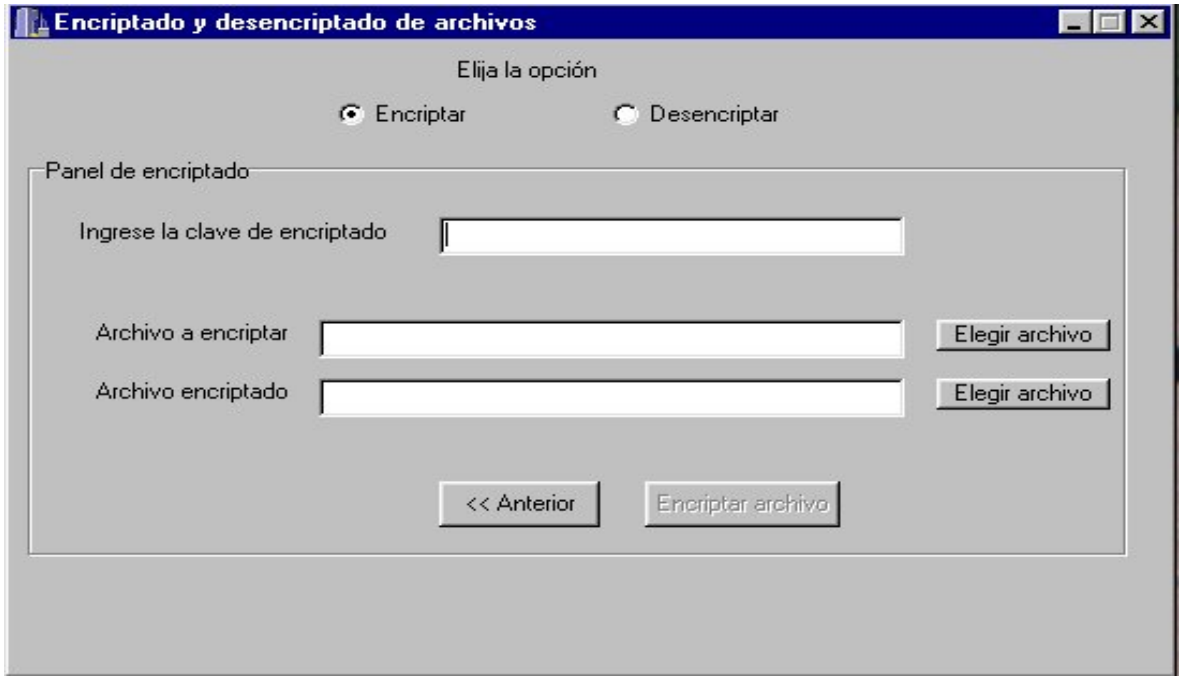

Imagen.2.Panel de encriptado

Como se dijo anteriormente, una vez que se presiona al botón **Siguiente** en el menú de presentación, aparece el formulario mostrado en la **Imagen.2**, donde el usuario puede elegir la operación a realizar, haciendo clic con el mouse en **Encriptar** o **Desencriptar**.

En el caso que el usuario desee encriptar, lo primero que debe ingresar en el panel es la clave de encriptado, la cual servirá luego para desencriptar. Una vez ingresada la misma, debe ser elegido el archivo a encriptar, con lo cual el usuario puede escribir la ruta de dicho archivo en la caja de texto que se encuentra al lado de la etiqueta **Archivo a encriptar**, o bien presionar el botón **Elegir archivo**, al lado de la caja de texto. Al presionar dicho botón, se abre un menú donde el usuario puede recorrer todos los directorios de su máquina, y elegir al archivo deseado a encriptar. Dicho menú es igual al utilizado en otros programas y se lo presenta a continuación en la **Imagen.3**:

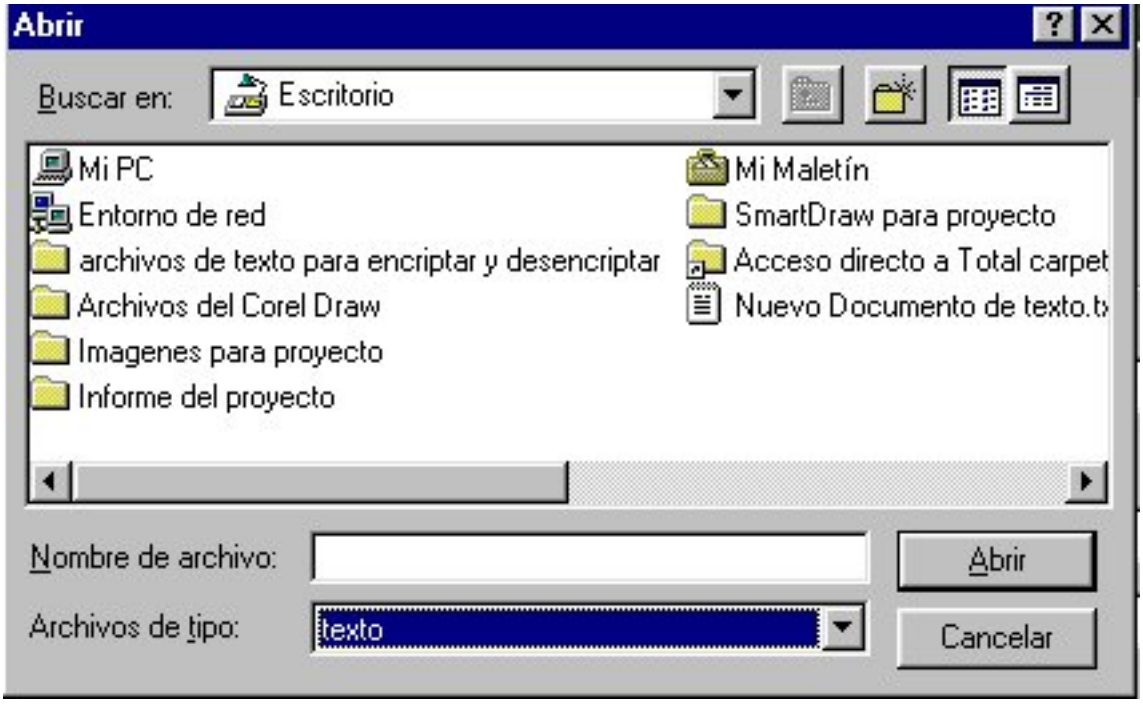

Imagen.3.Directorios de archivos

Dependiendo de que tipo sea el archivo a elegir, en la opción **Archivos de tipo**, el usuario puede elegir las siguientes opciones de tipos para ver en pantalla:

Archivos pdf (extensión **.pdf**).

Documentos de word (extensión **.doc**).

Páginas web (extensión **.htm**).

Documentos de texto (extensión **.txt**).

Todos los archivos (cualquier extensión).

Una vez que el usuario elige el archivo a encriptar mediante un clic con el mouse sobre el icono del mismo, aparece en la caja de texto que se encuentra al lado de la etiqueta **Nombre de archivo**, el nombre del archivo. Luego se presiona el botón **Abrir**, con lo cual en la caja de texto al lado de la etiqueta **Archivo a encriptar**, queda copiada la ruta del archivo, como se muestra a continuación en la **Imagen 4**:

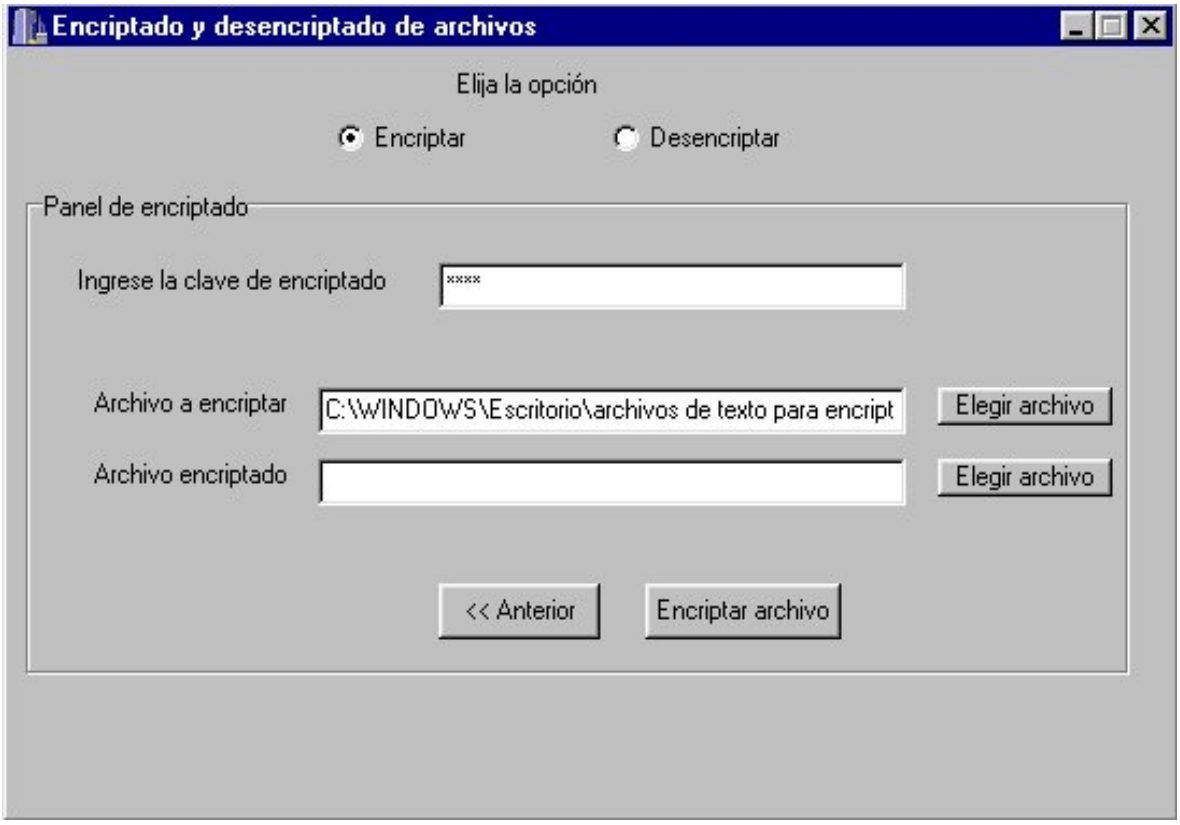

Imagen.4.Funcionamiento del botón Elegir archivo en la elección del archivo a encriptar

Una vez realizada la operación anterior, el usuario puede elegir el archivo y el directorio en el cual desea guardar al texto encriptado. Lo mismo puede ser realizado, escribiendo en la caja de texto que se encuentra al lado de la etiqueta **Archivo encriptado**, la ruta del archivo, o bien presionando el botón **Elegir archivo**, que se encuentra al lado de dicha caja de texto, con lo cual se abre el menú mostrado en la **Imagen 3**, y se realiza el mismo procedimiento explicado en la elección del archivo a encriptar, con lo cual en la caja de texto correspondiente a **Archivo encriptado** queda lo siguiente:

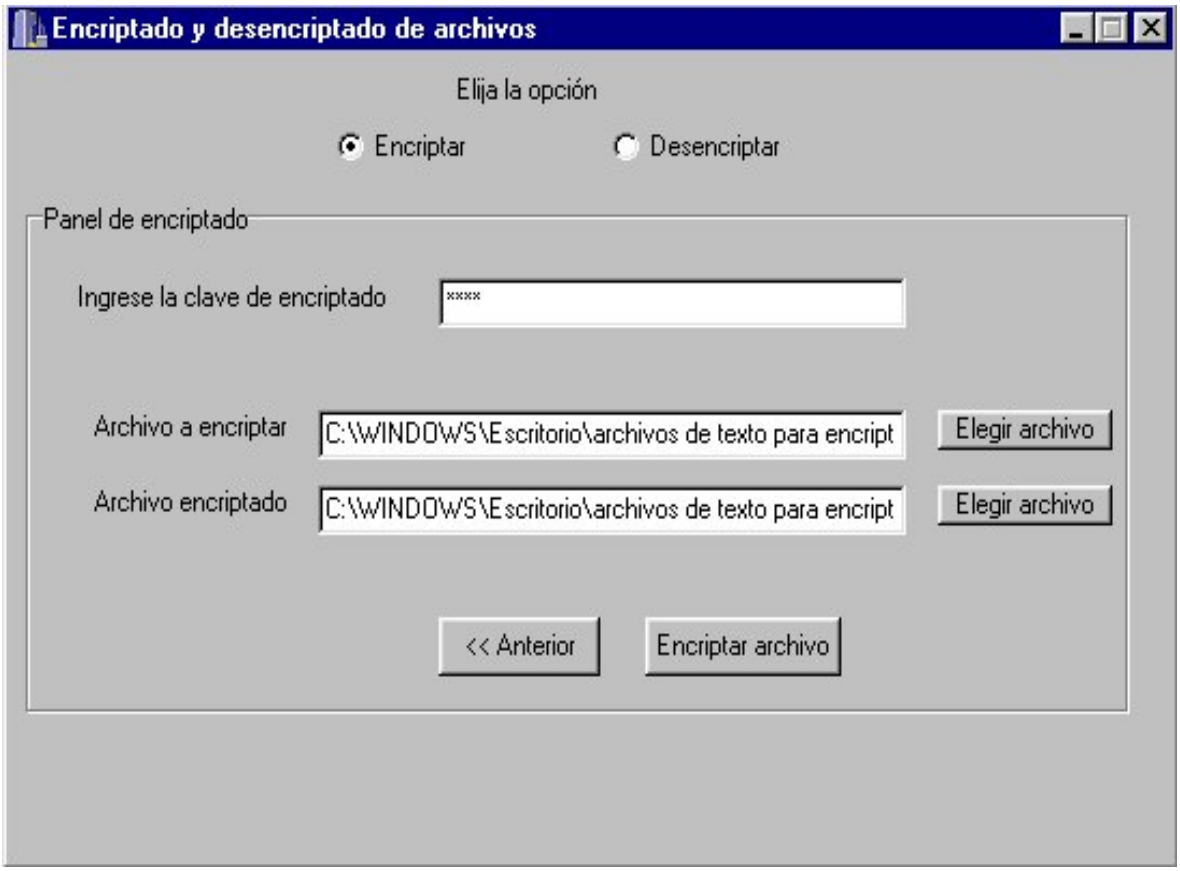

Imagen.5.Funcionamiento del botón Elegir archivo en la elección del archivo en el cual se guardará al texto encriptado

Como se observa en la **Imagen 5**, una vez que la ruta del archivo destino es copiada en la caja de texto, es habilitado el botón **Encriptar archivo**, con lo cual el usuario ahora puede presionar dicho botón para que se ejecute el encriptado. Cuando dicho botón es presionado, debajo del formulario aparece una barra de progreso, la cual luego de finalizar, aparece un cartel indicando que el encriptado ha finalizado.

Por último, faltaría explicar que presionando el botón **Anterior** mostrado en la **Imagen 4**, se retorna al menú de presentación mostrado en la **Imagen 1**.

5.2.2.Panel de desencriptado

En el caso que el usuario desee desencriptar, debe realizar un clic con el mouse en **Desencriptar**, con lo cual aparece el siguiente panel mostrado en la **Imagen 6**:

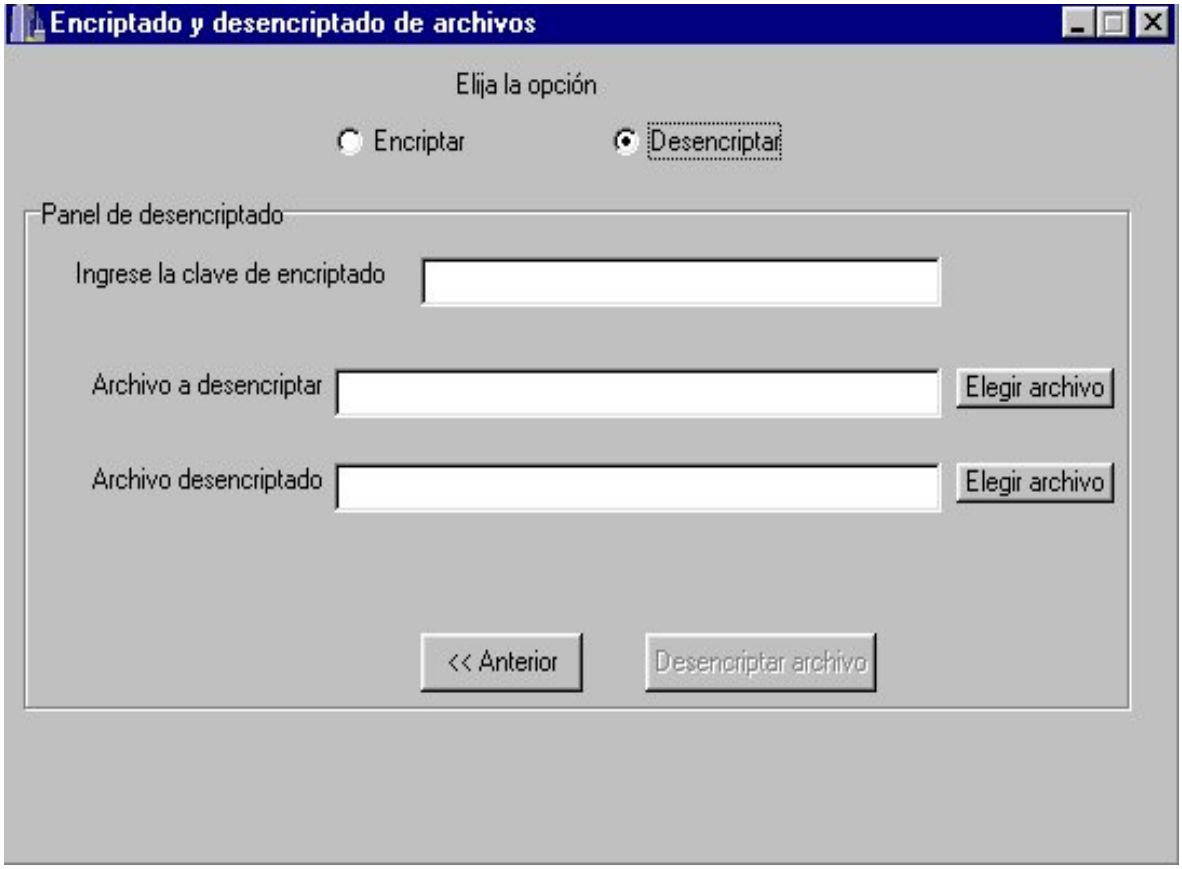

Imagen 6.Panel de desencriptado

Como en el caso del panel de encriptado, lo primero que el usuario debe ingresar es la misma clave ingresada en dicho panel, pues de lo contrario el archivo no podrá ser desencriptado. Una vez ingresada la misma, debe ser elegido el archivo a desencriptar, con lo cual el usuario puede escribir la ruta de dicho archivo en la caja de texto que se encuentra al lado de la etiqueta **Archivo a desencriptar**, o bien presionar el botón **Elegir archivo**, al lado de la caja de texto. Al presionar dicho botón, se abre el menú mostrado en la **Imagen 3**, y se realiza el mismo procedimiento explicado en la sección 5.2.1, con lo cual en la caja de texto al lado de la etiqueta **Archivo a desencriptar**, queda copiada la ruta del archivo, como se muestra a continuación en la **Imagen 7**:

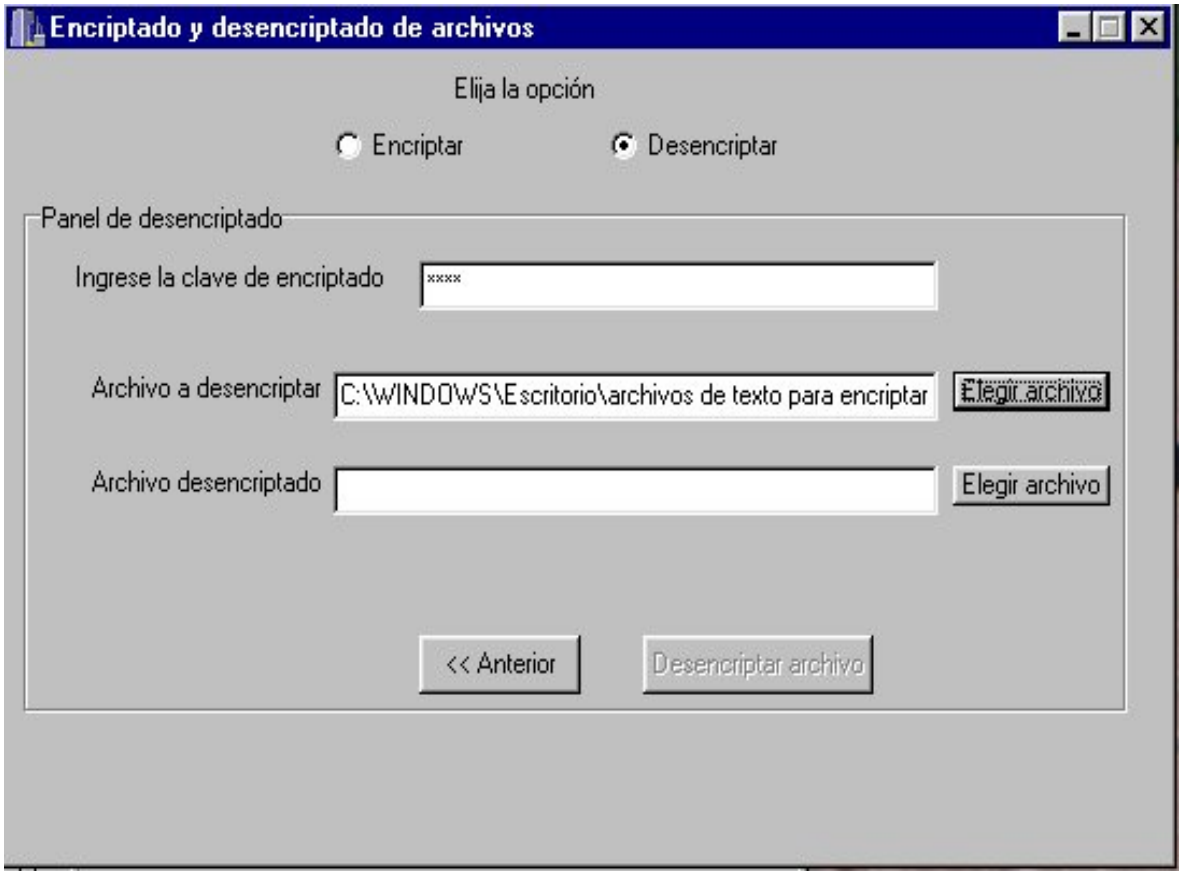

Imagen.7.Funcionamiento del botón Elegir archivo en la elección del archivo a desencriptar

Una vez realizada la operación anterior, el usuario puede elegir el archivo y el directorio en el cual desea guardar al texto desencriptado. Lo mismo puede ser realizado escribiendo en la caja de texto que se encuentra al lado de la etiqueta **Archivo desencriptado** la ruta del archivo, o bien presionando el botón **Elegir archivo**, que se encuentra al lado de dicha caja de texto, con lo cual se abre el menú mostrado en la **Imagen 3**, y se realiza el mismo procedimiento explicado en la sección 5.2.1, con lo cual en la caja de texto correspondiente a **Archivo desencriptado** queda lo siguiente:

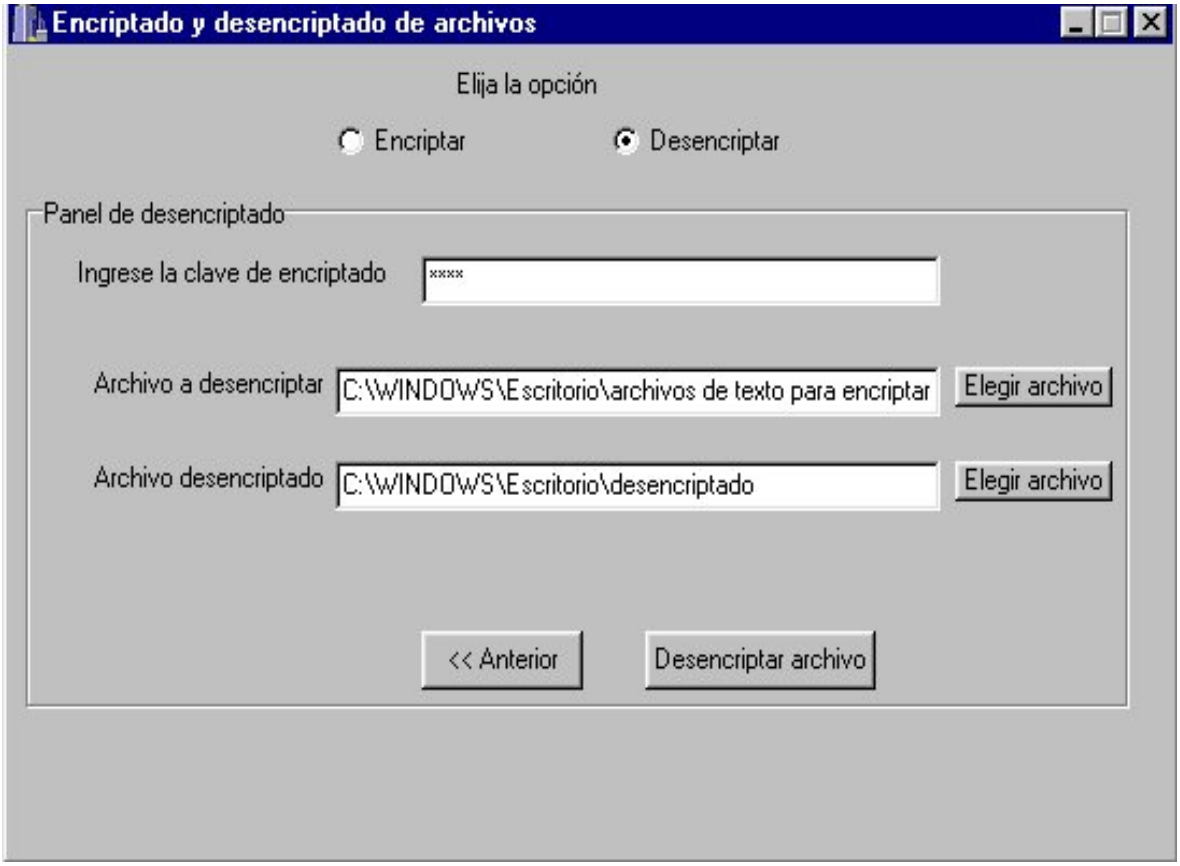

Imagen.8.Funcionamiento del botón Elegir archivo en la elección del archivo desencriptado

Como se observa en la **Imagen 8**, una vez que la ruta del archivo destino es copiada en la caja de texto, es habilitado el botón **Desencriptar archivo**, con lo cual el usuario ahora puede presionar dicho botón para que se ejecute el desencriptado. Cuando dicho botón es presionado, debajo del formulario aparece una barra de progreso, la cual luego de finalizar, aparece un cartel indicando que la operación ha finalizado. En este caso, también cabe destacar, que al ser presionado el botón de desencriptado, la extensión del archivo a encriptar será concatenada con el nombre del archivo destino, es decir, en el caso de la **Imagen 8**, al lado del nombre "desencriptado", será concatenada la extensión del archivo origen y el archivo desencriptado será creado con dicha extensión, con lo cual se obtendrá una réplica del archivo origen.

Por último, faltaría explicar que presionando el botón **Anterior** mostrado en la **Imagen 8**, se retorna al menú de presentación mostrado en la **Imagen 1**.

SECCIÓN VI

MEDICIONES DE VELOCIDADES DE ENCRIPTADO Y DESENCRIPTADO

6.Medición de velocidades de encriptado y desencriptado

Respecto a la velocidad del sistema, con fines de obtener valores de caracterización para el software diseñado, se realizaron pruebas sobre distintas pc's con diferentes sistemas operativos y velocidades de microprocesadores, obteniéndose los siguientes resultados:

Pentium III 800Mhz 512 MB Ram Win2K Pro SP2

Arch.	Ext.	Tamaño (bytes)	Tamaño (bits)	T.encriptado (seg)	Vel. encriptado (b.p.s.)	T.desencriptado (seg)	Vel. Desencriptado (b.p.s.)
1	jpg	102.379	819032	1	819032	2	409516
2	mht	223.089	1784712	2	892356	4	446178
3	pdf	452.318	3618544	3	1206181,333	7	516934,8571
4	exe	821.248	6569984	6	1094997,333	13	505383,3846
5	zip	1.156.528	9252224	8	1156528	18	514012,4444
6	pdf	2.066.618	16532944	14	1180924,571	33	500998,303
7	pdf	5.044.095	40352760	35	1152936	80	504409,5
8	mp3	8.274.818	66198544	56	1182116,857	132	501504,1212
9	zip	10.690.272	85522176	73	1171536,658	171	500129,6842
10	mpg	21.850.116	174800928	150	1165339,52	349	500862,2579
				Promedio:	1102194,827		489992,8553

Tabla 4. Medición de velocidades en una Pentium 3 800Mhz Win 2000 Pro

		Celeron 366MHZ	64MB RAM	Win98SE			
Arch	Ext.	Tamaño (bytes)	Tamaño (bits)	T.encriptado (seg)	Vel. encriptado (b.p.s.)	T.desencrip.(seg)	Vel. Desencrip. (b.p.s.)
1	jpg	102.379	819032	2	409516	5	163806,4
2	mht	223.089	1784712	5	356942,4	10	178471,2
3	pdf	452.318	3618544	9	402060,4444	20	180927,2
4	exe	821.248	6569984	17	386469,6471	36	182499,5556
5	zip	1.156.528	9252224	22	420555,6364	51	181416,1569
6	pdf	2.066.618	16532944	41	403242,5366	91	181680,7033
7	pdf	5.044.095	40352760	100	403527,6	224	180146,25
8	mp3	8.274.818	66198544	179	369824,2682	360	183884,8444
9	zip	10.690.272	85522176	211	405318,3697	459	186322,8235
10	mpg	21.850.116	174800928	433	403697,2933	957	182655,0972
				Promedio:	396115,4196		180181,0231

Tabla 5.Medición de velocidades en una Celeron 366Mhz Win 98

Celeron 366MHZ 64MB RAM Win 2K Pro SP2

Arch.	Ext.	Tamaño (bytes)	Tamaño (bits)	T.encriptado (seg)	Vel. encriptado (b.p.s.)	T.desencrip. (seg)	Vel. Desencrip.(b.p.s.)
1	jpg	102.379	819032	2	409516	4	204758
2	mht	223.089	1784712	4	446178	8	223089
3	pdf	452.318	3618544	7	516934,8571	17	212855,5294
4	exe	821.248	6569984	13	505383,3846	30	218999,4667
5	zip	1.156.528	9252224	18	514012,4444	43	215168
6	pdf	2.066.618	16532944	32	516654,5	77	214713,5584
7	pdf	5.044.095	40352760	81	498182,2222	190	212382,9474
8	mp3	8.274.818	66198544	130	509219,5692	308	214930,3377
9	zip	10.690.272	85522176	169	506048,3787	399	214341,2932
10	mpg	21.850.116	174800928	339	515636,9558	807	216605,8587
				Promedio:	493776,6312		214784,3992

Tabla 6. Medición de velocidades en una Celeron 366 Mhz Win 2000 Pro

		Pentium 233MHZ	32MB Ram	Win 95			
Arch.	Ext.	Tamaño (bytes)	Tamaño (bits)	T.encriptado (seg)	Vel. encriptado (b.p.s.)	T.Desencrip. (seg)	Vel. Desencrip. (b.p.s.)
1	jpg	102.379	819032	4	204758	11	74457,45455
2	mht	223.089	1784712	9	198301,3333	23	77596,17391
3	pdf	452.318	3618544	18	201030,2222	47	76990,29787
4	exe	821.248	6569984	33	199090,4242	87	75517,05747
5	zip	1.156.528	9252224	47	196855,8298	122	75837,90164
6	pdf	2.066.618	16532944	85	194505,2235	218	75839,19266
7	pdf	5.044.095	40352760	205	196842,7317	532	75851,05263
8	mp3	8.274.818	66198544	338	195853,6805	872	75915,76147
9	zip	10.690.272	85522176	434	197055,7051	1115	76701,50314
10	mpg	21.850.116	174800928	890	196405,5371	2259	77379,7822
				Promedio:	198069,8687		76208,61775

Tabla 7. Medición de velocidades en una Pentium 233 Mhz Win 95

En resumen, se obtuvo lo siguiente:

Equipo	Encriptado (bps)	Desencriptado (bps)
Pentium III 800MHZ 512MB Win 2K SP2	1102194,827	489992,8553
Celeron 366MHZ 64MB Win 2K	493776,6312	214784,3992
Celeron 366MHZ 64MB Win 98 SE	396115,4196	180181,0231
Pentium 233MHZ 32MB Win 95	198069,8687	76208,61775

Tabla 8. Comparación entre los distintos procesadores utilizados

Respecto a los resultados que se obtuvieron, se puede decir que la calidad del sistema es óptima en velocidad en los diferentes sistemas operativos y velocidades de microprocesadores citadas en esta sección. Para asegurar la perfomance del sistema en velocidad, es necesario compararlo con otros algoritmos de encriptado, dicha comparación fue realizada y los resultados fueron publicados en la segunda conferencia sobre el AES celebrada del 22 al 23 de Marzo 1999 en Roma, Italia, en la cual el enfoque primordial ha sido sobre la velocidad de los candidatos. Los candidatos más rápidos resultaron ser RC6, Twofish, Mars, Rijndael (A.E.S.) y E2. La velocidad de los algoritmos más veloces es realmente impresionante: dos ciclos del procesador para cada bit encriptado. En otras palabras en un Pentium de 300 MHz (un procesador de mediano tamaño hoy que seguramente se superará en los próximos años) se pueden encriptar 18 MBytes por segundo!. Se considera que una comunicación telefónica digital de alta calidad requiere menos de 2 KBytes por segundo, lo que significa que una PC puede encriptar hoy más de 9 mil llamadas telefónicas concurrentemente. [1]

A continuación se presenta un cuadro comparativo de los algoritmos en la Tabla 9.

[1] A.E.S, El algoritmo de encriptación del próximo siglo.

Algoritmo	Longitud de la clave	Rating
Blowfish	1-418 bits	Segundo puesto
D.E.S.	56 bits	Cuarto puesto
I.D.E.A.	128 bits	Segundo puesto
M.A.R.S. *	128-256 bits	Tercer puesto
RC2	1-2048 bits	Primer puesto
RC4	1-2048 bits	Segundo puesto
RC5	128-256 bits	Tercer puesto
RC6 *	128-256 bits	Tercer puesto
Rijndael (A.E.S.) *	128-256 bits	Primer puesto
Serpent *	128-256 bits	Tercer puesto
Triple-D.E.S.	168 bits	Segundo puesto
Twofish	128-256 bits	Tercer puesto

Tabla 9. Comparación de algoritmos criptográficos [2]

*Algoritmos que fueron finalistas en la conferencia sobre el A.E.S, de la cual resultó ganador el algoritmo Rijndael.

Como se observa en la Tabla 9, los algoritmos fueron clasificados por orden de mérito, desde los mejores algoritmos, hasta los algoritmos que no son muy recomendados, debido a la corta longitud de la clave, como es el caso del D.E.S., cuya longitud es de 56 bits, que comparándola con el A.E.S., las diferencias son notorias, debido a que la cantidad de posibles combinaciones de claves a obtener, depende de la longitud de la misma; con lo cual las probabilidades de que un ataque sea exitoso aumentan, mientras más corta sea la clave.

Volviendo al tema puntual de este algoritmo de encriptado en particular, sus velocidades promedio de encriptado y desencriptado que figuran en la Tabla 8 fueron

[2] Presentación en pdf de D.E.S.-R.S.A.-A.E.S. .

comparadas con otras implementaciones del A.E.S., cuyos resultados se presentan a continuación.

La siguiente tabla muestra las velocidades de encriptado y desencriptado de otra implementación del A.E.S., las pruebas fueron realizadas utilizando una clave de 128 bits, en una Pentium Pro de 200Mhz:

ALGORITMO	RIJNDAEL
VELOCIDAD DE ENCRIPTADO PROMEDIO (MBITS/SEG)	68.4
VELOCIDAD DE DESENCRIPTADO PROMEDIO (MBITS/SEG)	72.7

Tabla 10. Velocidades de encriptado y desencriptado con otra implementación del A.E.S.[3]

Con otra implementación distinta a la anterior del Rijndael en la cual también se utilizó una clave de 128 bits y se realizaron las pruebas en una Pentium Pro de 200Mhz, se obtuvieron los siguientes resultados:

ALGORITMO	RIJNDAEL
VELOCIDAD DE ENCRIPTADO PROMEDIO (MBITS/SEG)	69.0
VELOCIDAD DE DESENCRIPTADO PROMEDIO (MBITS/SEG)	68.6

Tabla 11. Velocidades de encriptado y desencriptado con otra implementación del A.E.S.[4]

Comparando las Tablas 10 y 11 con la Tabla 8, se observa que si bien en esta última, las velocidades son más lentas, hay que tener en cuenta que la velocidad depende tanto del

[3] http://fp.gladman.plus.com/cryptography_technology/aesr1/index.htm

[4] http://fp.gladman.plus.com/cryptography_technology/aesr1/index.htm

algoritmo, como de la implementación, el lenguaje de programación, etc; de hecho se pudo comprobar que la implementación que figura en la Tabla 8 varía en velocidad en la misma máquina con diferentes sistemas operativos.

SECCIÓN VII

CONCLUSIONES

7.Conclusiones

Se ha logrado la obtención de un software de encriptado y desencriptado de sencillo manejo, basado en uno de los algoritmos recientemente elegidos como estándar que utiliza transformaciones matemáticas de una cierta complejidad en el campo finito GF(2^8).

El software fue realizado en C++Builder, por ser el lenguaje C uno de los lenguajes más potentes y con mayor capacidad en el manejo de memoria y de interrupciones que otros programas.

Respecto a la velocidad del sistema, se puede decir que la calidad del mismo es óptima en los diferentes sistemas operativos y velocidades de microprocesadores citadas en la sección VI (Pentium III 800Mhz Windows 2000 Pro, Celeron 366Mhz Windows 98, Celeron 366Mhz Windows 2000 Pro, Pentium 233Mhz Windows 95). Para asegurar que el sistema es óptimo en velocidad, es necesario compararlo con otros algoritmos de encriptado, dicha comparación fue realizada y los resultados fueron publicados en la segunda conferencia sobre el AES celebrada del 22 al 23 de Marzo 1999 en Roma, Italia, en la cual el enfoque primordial ha sido sobre la velocidad de los candidatos. Los candidatos más rápidos resultaron ser RC6, Twofish, Mars, Rijndael (A.E.S.) y E2.

En cuanto a las dificultades encontradas, se puede mencionar que el punto que más problemas presentó, es resolver el tema de cómo desencriptar correctamente al archivo, pues como se toman bloques de 16 bytes para decodificarlos, pueden ser guardados en el archivo destino caracteres adicionales, sobre todo cuando se llega al final del archivo encriptado, pues dentro del bloque de 16 bytes estaría el fin de archivo, con lo cual se desencriptarían caracteres adicionales que serían basura o ruido. Por este motivo se pensó en encriptar al tamaño del archivo para luego desencriptarlo y almacenarlo en un contador, el cual luego es decrementado en una unidad a medida que un carácter desencriptado es guardado en el archivo destino, pues si el contador no existiera, caracteres adicionales (ruido) serían desencriptados y guardados en el archivo destino erróneamente. Entre otras dificultades, se puede mencionar que las funciones del soft que resultaron más complicadas para programar, fueron sobre todo MixColumns y su inversa y la Expansión de la clave, las cuales requieren de numerosas y complejas operaciones matemáticas.

Entre las posibles mejoras futuras que se le puede realizar al soft, se puede mencionar el uso de librerías, lo cual facilitaría el manejo y el mantenimiento del programa. Mantenimiento se refiere a que el uso de las librerías, facilita la corrección del mismo así como también permite que el programa pueda ser compilado con otro lenguaje de programación que no sea necesariamente C++. En cuanto al manejo por parte del usuario, se

puede facilitar todavía más el manejo de la interfaz gráfica, sobre todo para el uso de correo electrónico, mediante la implementación de un “plug-in” del programa de correo. El mismo es un programa que se “enchufa” en el outlook y le agrega las facilidades que uno desee. En este caso, lo que se puede hacer es un “plug-in” que encripte y desencripte los mensajes en forma transparente, es decir, sin que el usuario se de cuenta.

Por último, habría que analizar las ventajas del A.E.S. sobre su antecesor, el D.E.S., una de las principales sería la longitud de la clave (cuanto más grande es la longitud de la clave, más posibles combinaciones se pueden formar), pues este último utiliza una clave de 56 bits (el Triple D.E.S. una clave de 168 bits) y el A.E.S., una clave de 128 bits, 192 ó 256 bits, lo cual significa que este último algoritmo es más dificil de desencriptar utilizando el ataque de la “fuerza bruta”, o cualquier otro tipo de ataque, lo cual no quiere decir que sea imposible realizarlo;pues en criptografía nada es imposible, pero sería más dificultoso. También una de las desventajas que presenta el D.E.S. sobre el A.E.S., es el hecho de la existencia de una “puerta trasera”, que permite quebrar a las comunicaciones encriptadas y el hecho de que sea más lento que el A.E.S.

SECCIÓN VIII

BIBLIOGRAFÍA

8.Bibliografía

1.The Rijndael Block Cipher A.E.S. Proposal.

Autores: Joan Daemen y Vicent Rijmen.

2.Algorithm A.E.S.-Rijndael.

Autor: José de Jesús Angel.

3. Specification for the Advanced Encryption Standard (A.E.S).

Autor: Federal Information Processing Standards Publication 197.

4.A.E.S. El algoritmo de encriptación del próximo siglo.

SECCIÓN IX

ANEXO A

CÓDIGO FUENTE DEL PROGRAMA CORRESPONDIENTE A LAS FUNCIONES DE ENCRIPTADO Y DESENCRIPTADO

El siguiente es el código fuente correspondiente al archivo de extensión .h, que contiene a las funciones de encriptado y desencriptado.

```
int sboxtransf[256];
int vector1[4];
int sboxtransfinv[256];

 int llavedeencriptado( int llave[][4],int n1,int reserva[][4],int n2,AnsiString vectordeclave)

{

  unsigned char *vector2;
  int i,s,m;



    vector2=vectordeclave.c_str();
    i=0;

    for (m=0; m<4; m++)
     for (s=0; s<4; s++)
     {
        if (i<=sizeof(vector2))
        {
           llave[s][m]=vector2[i];
           reserva[s][m]=vector2[i];
           i++;
           if (i==16)
             break;
        }
        else
        {
           llave[s][m]='\0';
           reserva[s][m]='\0';
        }
     }
```

```
}


int matrices2(int sboxtransf[],int num1,int vector1[],int num3)
{
  int a;
  int b,c;
  int d;
  int e,f;

      sboxtransf[0]=0x63;
      sboxtransf[1]=0x7c;
      sboxtransf[2]=0x77;
      sboxtransf[3]=0x7b;
      sboxtransf[4]=0xf2;
      sboxtransf[5]=0x6b;
      sboxtransf[6]=0x6f;
      sboxtransf[7]=0xc5;
      sboxtransf[8]=0x30;
      sboxtransf[9]=0x01;
      sboxtransf[10]=0x67;
      sboxtransf[11]=0x2b;
      sboxtransf[12]=0xfe;
      sboxtransf[13]=0xd7;
      sboxtransf[14]=0xab;
      sboxtransf[15]=0x76;
      sboxtransf[16]=0xca;
      sboxtransf[17]=0x82;
      sboxtransf[18]=0xc9;
      sboxtransf[19]=0x7d;
      sboxtransf[20]=0xfa;
      sboxtransf[21]=0x59;
      sboxtransf[22]=0x47;
      sboxtransf[23]=0xf0;
      sboxtransf[24]=0xad;
      sboxtransf[25]=0xd4;
      sboxtransf[26]=0xa2;
```

```
sboxtransf[27]=0xaf;
sboxtransf[28]=0x9c;
sboxtransf[29]=0xa4;
sboxtransf[30]=0x72;
sboxtransf[31]=0xc0;
sboxtransf[32]=0xb7;
sboxtransf[33]=0xfd;
sboxtransf[34]=0x93;
sboxtransf[35]=0x26;
sboxtransf[36]=0x36;
sboxtransf[37]=0x3f;
sboxtransf[38]=0xf7;
sboxtransf[39]=0xcc;
sboxtransf[40]=0x34;
sboxtransf[41]=0xa5;
sboxtransf[42]=0xe5;
sboxtransf[43]=0xf1;
sboxtransf[44]=0x71;
sboxtransf[45]=0xd8;
sboxtransf[46]=0x31;
sboxtransf[47]=0x15;
sboxtransf[48]=0x04;
sboxtransf[49]=0xc7;
sboxtransf[50]=0x23;
sboxtransf[51]=0xc3;
sboxtransf[52]=0x18;
sboxtransf[53]=0x96;
sboxtransf[54]=0x05;
sboxtransf[55]=0x9a;
sboxtransf[56]=0x07;
sboxtransf[57]=0x12;
sboxtransf[58]=0x80;
sboxtransf[59]=0xe2;
sboxtransf[60]=0xeb;
sboxtransf[61]=0x27;
sboxtransf[62]=0xb2;
sboxtransf[63]=0x75;
```

```
sboxtransf[64]=0x09;
sboxtransf[65]=0x83;
sboxtransf[66]=0x2c;
sboxtransf[67]=0x1a;
sboxtransf[68]=0x1b;
sboxtransf[69]=0x6e;
sboxtransf[70]=0x5a;
sboxtransf[71]=0xa0;
sboxtransf[72]=0x52;
sboxtransf[73]=0x3b;
sboxtransf[74]=0xd6;
sboxtransf[75]=0xb3;
sboxtransf[76]=0x29;
sboxtransf[77]=0xe3;
sboxtransf[78]=0x2f;
sboxtransf[79]=0x84;
sboxtransf[80]=0x53;
sboxtransf[81]=0xd1;
sboxtransf[82]=0x00;
sboxtransf[83]=0xed;
sboxtransf[84]=0x20;
sboxtransf[85]=0xfc;
sboxtransf[86]=0xb1;
sboxtransf[87]=0x5b;
sboxtransf[88]=0x6a;
sboxtransf[89]=0xcb;
sboxtransf[90]=0xbe;
sboxtransf[91]=0x39;
sboxtransf[92]=0x4a;
sboxtransf[93]=0x4c;
sboxtransf[94]=0x58;
sboxtransf[95]=0xcf;
sboxtransf[96]=0xd0;
sboxtransf[97]=0xef;
sboxtransf[98]=0xaa;
sboxtransf[99]=0xfb;
sboxtransf[100]=0x43;
```

```
sboxtransf[101]=0x4d;
sboxtransf[102]=0x33;
sboxtransf[103]=0x85;
sboxtransf[104]=0x45;
sboxtransf[105]=0xf9;
sboxtransf[106]=0x02;
sboxtransf[107]=0x7f;
sboxtransf[108]=0x50;
sboxtransf[109]=0x3c;
sboxtransf[110]=0x9f;
sboxtransf[111]=0xa8;
sboxtransf[112]=0x51;
sboxtransf[113]=0xa3;
sboxtransf[114]=0x40;
sboxtransf[115]=0x8f;
sboxtransf[116]=0x92;
sboxtransf[117]=0x9d;
sboxtransf[118]=0x38;
sboxtransf[119]=0xf5;
sboxtransf[120]=0xbc;
sboxtransf[121]=0xb6;
sboxtransf[122]=0xda;
sboxtransf[123]=0x21;
sboxtransf[124]=0x10;
sboxtransf[125]=0xff;
sboxtransf[126]=0xf3;
sboxtransf[127]=0xd2;
sboxtransf[128]=0xcd;
sboxtransf[129]=0x0c;
sboxtransf[130]=0x13;
sboxtransf[131]=0xec;
sboxtransf[132]=0x5f;
sboxtransf[133]=0x97;
sboxtransf[134]=0x44;
sboxtransf[135]=0x17;
sboxtransf[136]=0xc4;
sboxtransf[137]=0xa7;
```

```
sboxtransf[138]=0x7e;
sboxtransf[139]=0x3d;
sboxtransf[140]=0x64;
sboxtransf[141]=0x5d;
sboxtransf[142]=0x19;
sboxtransf[143]=0x73;
sboxtransf[144]=0x60;
sboxtransf[145]=0x81;
sboxtransf[146]=0x4f;
sboxtransf[147]=0xdc;
sboxtransf[148]=0x22;
sboxtransf[149]=0x2a;
sboxtransf[150]=0x90;
sboxtransf[151]=0x88;
sboxtransf[152]=0x46;
sboxtransf[153]=0xee;
sboxtransf[154]=0xb8;
sboxtransf[155]=0x14;
sboxtransf[156]=0xde;
sboxtransf[157]=0x5e;
sboxtransf[158]=0x0b;
sboxtransf[159]=0xdb;
sboxtransf[160]=0xe0;
sboxtransf[161]=0x32;
sboxtransf[162]=0x3a;
sboxtransf[163]=0x0a;
sboxtransf[164]=0x49;
sboxtransf[165]=0x06;
sboxtransf[166]=0x24;
sboxtransf[167]=0x5c;
sboxtransf[168]=0xc2;
sboxtransf[169]=0xd3;
sboxtransf[170]=0xac;
sboxtransf[171]=0x62;
sboxtransf[172]=0x91;
sboxtransf[173]=0x95;
sboxtransf[174]=0xe4;
```

```
sboxtransf[175]=0x79;
sboxtransf[176]=0xe7;
sboxtransf[177]=0xc8;
sboxtransf[178]=0x37;
sboxtransf[179]=0x6d;
sboxtransf[180]=0x8d;
sboxtransf[181]=0xd5;
sboxtransf[182]=0x4e;
sboxtransf[183]=0xa9;
sboxtransf[184]=0x6c;
sboxtransf[185]=0x56;
sboxtransf[186]=0xf4;
sboxtransf[187]=0xea;
sboxtransf[188]=0x65;
sboxtransf[189]=0x7a;
sboxtransf[190]=0xae;
sboxtransf[191]=0x08;
sboxtransf[192]=0xba;
sboxtransf[193]=0x78;
sboxtransf[194]=0x25;
sboxtransf[195]=0x2e;
sboxtransf[196]=0x1c;
sboxtransf[197]=0xa6;
sboxtransf[198]=0xb4;
sboxtransf[199]=0xc6;
sboxtransf[200]=0xe8;
sboxtransf[201]=0xdd;
sboxtransf[202]=0x74;
sboxtransf[203]=0x1f;
sboxtransf[204]=0x4b;
sboxtransf[205]=0xbd;
sboxtransf[206]=0x8b;
sboxtransf[207]=0x8a;
sboxtransf[208]=0x70;
sboxtransf[209]=0x3e;
sboxtransf[210]=0xb5;
sboxtransf[211]=0x66;
```

```
sboxtransf[212]=0x48;
sboxtransf[213]=0x03;
sboxtransf[214]=0xf6;
sboxtransf[215]=0x0e;
sboxtransf[216]=0x61;
sboxtransf[217]=0x35;
sboxtransf[218]=0x57;
sboxtransf[219]=0xb9;
sboxtransf[220]=0x86;
sboxtransf[221]=0xc1;
sboxtransf[222]=0x1d;
sboxtransf[223]=0x9e;
sboxtransf[224]=0xe1;
sboxtransf[225]=0xf8;
sboxtransf[226]=0x98;
sboxtransf[227]=0x11;
sboxtransf[228]=0x69;
sboxtransf[229]=0xd9;
sboxtransf[230]=0x8e;
sboxtransf[231]=0x94;
sboxtransf[232]=0x9b;
sboxtransf[233]=0x1e;
sboxtransf[234]=0x87;
sboxtransf[235]=0xe9;
sboxtransf[236]=0xce;
sboxtransf[237]=0x55;
sboxtransf[238]=0x28;
sboxtransf[239]=0xdf;
sboxtransf[240]=0x8c;
sboxtransf[241]=0xa1;
sboxtransf[242]=0x89;
sboxtransf[243]=0x0d;
sboxtransf[244]=0xbf;
sboxtransf[245]=0xe6;
sboxtransf[246]=0x42;
sboxtransf[247]=0x68;
sboxtransf[248]=0x41;
```

```
        sboxtransf[249]=0x99;
        sboxtransf[250]=0x2d;
        sboxtransf[251]=0x0f;
        sboxtransf[252]=0xb0;
        sboxtransf[253]=0x54;
        sboxtransf[254]=0xbb;
        sboxtransf[255]=0x16;



         vector1[0]=0x01;
         vector1[1]=0x00;
         vector1[2]=0x00;
         vector1[3]=0x00;

       }

int matrices(int sboxtransfinv[],int num5)
{

   int g;

          sboxtransfinv[0]=0x52;
        sboxtransfinv[1]=0x09;
        sboxtransfinv[2]=0x6a;
        sboxtransfinv[3]=0xd5;
        sboxtransfinv[4]=0x30;
        sboxtransfinv[5]=0x36;
        sboxtransfinv[6]=0xa5;
        sboxtransfinv[7]=0x38;
        sboxtransfinv[8]=0xbf;
        sboxtransfinv[9]=0x40;
        sboxtransfinv[10]=0xa3;
        sboxtransfinv[11]=0x9e;
        sboxtransfinv[12]=0x81;
        sboxtransfinv[13]=0xf3;
        sboxtransfinv[14]=0xd7;
```

```
sboxtransfinv[15]=0xfb;
sboxtransfinv[16]=0x7c;
sboxtransfinv[17]=0xe3;
sboxtransfinv[18]=0x39;
sboxtransfinv[19]=0x82;
sboxtransfinv[20]=0x9b;
sboxtransfinv[21]=0x2f;
sboxtransfinv[22]=0xff;
sboxtransfinv[23]=0x87;
sboxtransfinv[24]=0x34;
sboxtransfinv[25]=0x8e;
sboxtransfinv[26]=0x43;
sboxtransfinv[27]=0x44;
sboxtransfinv[28]=0xc4;
sboxtransfinv[29]=0xde;
sboxtransfinv[30]=0xe9;
sboxtransfinv[31]=0xcb;
sboxtransfinv[32]=0x54;
sboxtransfinv[33]=0x7b;
sboxtransfinv[34]=0x94;
sboxtransfinv[35]=0x32;
sboxtransfinv[36]=0xa6;
sboxtransfinv[37]=0xc2;
sboxtransfinv[38]=0x23;
sboxtransfinv[39]=0x3d;
sboxtransfinv[40]=0xee;
sboxtransfinv[41]=0x4c;
sboxtransfinv[42]=0x95;
sboxtransfinv[43]=0x0b;
sboxtransfinv[44]=0x42;
sboxtransfinv[45]=0xfa;
sboxtransfinv[46]=0xc3;
sboxtransfinv[47]=0x4e;
sboxtransfinv[48]=0x08;
sboxtransfinv[49]=0x2e;
sboxtransfinv[50]=0xa1;
sboxtransfinv[51]=0x66;
```

```
sboxtransfinv[52]=0x28;
sboxtransfinv[53]=0xd9;
sboxtransfinv[54]=0x24;
sboxtransfinv[55]=0xb2;
sboxtransfinv[56]=0x76;
sboxtransfinv[57]=0x5b;
sboxtransfinv[58]=0xa2;
sboxtransfinv[59]=0x49;
sboxtransfinv[60]=0x6d;
sboxtransfinv[61]=0x8b;
sboxtransfinv[62]=0xd1;
sboxtransfinv[63]=0x25;
sboxtransfinv[64]=0x72;
sboxtransfinv[65]=0xf8;
sboxtransfinv[66]=0xf6;
sboxtransfinv[67]=0x64;
sboxtransfinv[68]=0x86;
sboxtransfinv[69]=0x68;
sboxtransfinv[70]=0x98;
sboxtransfinv[71]=0x16;
sboxtransfinv[72]=0xd4;
sboxtransfinv[73]=0xa4;
sboxtransfinv[74]=0x5c;
sboxtransfinv[75]=0xcc;
sboxtransfinv[76]=0x5d;
sboxtransfinv[77]=0x65;
sboxtransfinv[78]=0xb6;
sboxtransfinv[79]=0x92;
sboxtransfinv[80]=0x6c;
sboxtransfinv[81]=0x70;
sboxtransfinv[82]=0x48;
sboxtransfinv[83]=0x50;
sboxtransfinv[84]=0xfd;
sboxtransfinv[85]=0xed;
sboxtransfinv[86]=0xb9;
sboxtransfinv[87]=0xda;
sboxtransfinv[88]=0x5e;
```

```
sboxtransfinv[89]=0x15;
sboxtransfinv[90]=0x46;
sboxtransfinv[91]=0x57;
sboxtransfinv[92]=0xa7;
sboxtransfinv[93]=0x8d;
sboxtransfinv[94]=0x9d;
sboxtransfinv[95]=0x84;
sboxtransfinv[96]=0x90;
sboxtransfinv[97]=0xd8;
sboxtransfinv[98]=0xab;
sboxtransfinv[99]=0x00;
sboxtransfinv[100]=0x8c;
sboxtransfinv[101]=0xbc;
sboxtransfinv[102]=0xd3;
sboxtransfinv[103]=0x0a;
sboxtransfinv[104]=0xf7;
sboxtransfinv[105]=0xe4;
sboxtransfinv[106]=0x58;
sboxtransfinv[107]=0x05;
sboxtransfinv[108]=0xb8;
sboxtransfinv[109]=0xb3;
sboxtransfinv[110]=0x45;
sboxtransfinv[111]=0x06;
sboxtransfinv[112]=0xd0;
sboxtransfinv[113]=0x2c;
sboxtransfinv[114]=0x1e;
sboxtransfinv[115]=0x8f;
sboxtransfinv[116]=0xca;
sboxtransfinv[117]=0x3f;
sboxtransfinv[118]=0x0f;
sboxtransfinv[119]=0x02;
sboxtransfinv[120]=0xc1;
sboxtransfinv[121]=0xaf;
sboxtransfinv[122]=0xbd;
sboxtransfinv[123]=0x03;
sboxtransfinv[124]=0x01;
sboxtransfinv[125]=0x13;
```

```
sboxtransfinv[126]=0x8a;
sboxtransfinv[127]=0x6b;
sboxtransfinv[128]=0x3a;
sboxtransfinv[129]=0x91;
sboxtransfinv[130]=0x11;
sboxtransfinv[131]=0x41;
sboxtransfinv[132]=0x4f;
sboxtransfinv[133]=0x67;
sboxtransfinv[134]=0xdc;
sboxtransfinv[135]=0xea;
sboxtransfinv[136]=0x97;
sboxtransfinv[137]=0xf2;
sboxtransfinv[138]=0xcf;
sboxtransfinv[139]=0xce;
sboxtransfinv[140]=0xf0;
sboxtransfinv[141]=0xb4;
sboxtransfinv[142]=0xe6;
sboxtransfinv[143]=0x73;
sboxtransfinv[144]=0x96;
sboxtransfinv[145]=0xac;
sboxtransfinv[146]=0x74;
sboxtransfinv[147]=0x22;
sboxtransfinv[148]=0xe7;
sboxtransfinv[149]=0xad;
sboxtransfinv[150]=0x35;
sboxtransfinv[151]=0x85;
sboxtransfinv[152]=0xe2;
sboxtransfinv[153]=0xf9;
sboxtransfinv[154]=0x37;
sboxtransfinv[155]=0xe8;
sboxtransfinv[156]=0x1c;
sboxtransfinv[157]=0x75;
sboxtransfinv[158]=0xdf;
sboxtransfinv[159]=0x6e;
sboxtransfinv[160]=0x47;
sboxtransfinv[161]=0xf1;
sboxtransfinv[162]=0x1a;
```

```
sboxtransfinv[163]=0x71;
sboxtransfinv[164]=0x1d;
sboxtransfinv[165]=0x29;
sboxtransfinv[166]=0xc5;
sboxtransfinv[167]=0x89;
sboxtransfinv[168]=0x6f;
sboxtransfinv[169]=0xb7;
sboxtransfinv[170]=0x62;
sboxtransfinv[171]=0x0e;
sboxtransfinv[172]=0xaa;
sboxtransfinv[173]=0x18;
sboxtransfinv[174]=0xbe;
sboxtransfinv[175]=0x1b;
sboxtransfinv[176]=0xfc;
sboxtransfinv[177]=0x56;
sboxtransfinv[178]=0x3e;
sboxtransfinv[179]=0x4b;
sboxtransfinv[180]=0xc6;
sboxtransfinv[181]=0xd2;
sboxtransfinv[182]=0x79;
sboxtransfinv[183]=0x20;
sboxtransfinv[184]=0x9a;
sboxtransfinv[185]=0xdb;
sboxtransfinv[186]=0xc0;
sboxtransfinv[187]=0xfe;
sboxtransfinv[188]=0x78;
sboxtransfinv[189]=0xcd;
sboxtransfinv[190]=0x5a;
sboxtransfinv[191]=0xf4;
sboxtransfinv[192]=0x1f;
sboxtransfinv[193]=0xdd;
sboxtransfinv[194]=0xa8;
sboxtransfinv[195]=0x33;
sboxtransfinv[196]=0x88;
sboxtransfinv[197]=0x07;
sboxtransfinv[198]=0xc7;
sboxtransfinv[199]=0x31;
```

```
sboxtransfinv[200]=0xb1;
sboxtransfinv[201]=0x12;
sboxtransfinv[202]=0x10;
sboxtransfinv[203]=0x59;
sboxtransfinv[204]=0x27;
sboxtransfinv[205]=0x80;
sboxtransfinv[206]=0xec;
sboxtransfinv[207]=0x5f;
sboxtransfinv[208]=0x60;
sboxtransfinv[209]=0x51;
sboxtransfinv[210]=0x7f;
sboxtransfinv[211]=0xa9;
sboxtransfinv[212]=0x19;
sboxtransfinv[213]=0xb5;
sboxtransfinv[214]=0x4a;
sboxtransfinv[215]=0x0d;
sboxtransfinv[216]=0x2d;
sboxtransfinv[217]=0xe5;
sboxtransfinv[218]=0x7a;
sboxtransfinv[219]=0x9f;
sboxtransfinv[220]=0x93;
sboxtransfinv[221]=0xc9;
sboxtransfinv[222]=0x9c;
sboxtransfinv[223]=0xef;
sboxtransfinv[224]=0xa0;
sboxtransfinv[225]=0xe0;
sboxtransfinv[226]=0x3b;
sboxtransfinv[227]=0x4d;
sboxtransfinv[228]=0xae;
sboxtransfinv[229]=0x2a;
sboxtransfinv[230]=0xf5;
sboxtransfinv[231]=0xb0;
sboxtransfinv[232]=0xc8;
sboxtransfinv[233]=0xeb;
sboxtransfinv[234]=0xbb;
sboxtransfinv[235]=0x3c;
sboxtransfinv[236]=0x83;
```

```
    sboxtransfinv[237]=0x53;
    sboxtransfinv[238]=0x99;
    sboxtransfinv[239]=0x61;
    sboxtransfinv[240]=0x17;
    sboxtransfinv[241]=0x2b;
    sboxtransfinv[242]=0x04;
    sboxtransfinv[243]=0x7e;
    sboxtransfinv[244]=0xba;
    sboxtransfinv[245]=0x77;
    sboxtransfinv[246]=0xd6;
    sboxtransfinv[247]=0x26;
    sboxtransfinv[248]=0xe1;
    sboxtransfinv[249]=0x69;
    sboxtransfinv[250]=0x14;
    sboxtransfinv[251]=0x63;
    sboxtransfinv[252]=0x55;
    sboxtransfinv[253]=0x21;
    sboxtransfinv[254]=0x0c;
    sboxtransfinv[255]=0x7d;




  }
/*Funciones del programa de desencriptado */
        unsigned char matriz_caracteres(unsigned char matriz_char[][4],int numero0,
int num[][4],int numero1)
 {
   /* Armado de la matriz en caracteres*/
    int y,k;
    int s,m;

    s=0;
    m=0;
    k=0;
    for (y=0; y<4; y++)
    {
```

```
    matriz_char[s][m]=num[y][k];
    s++;
  }


  s=0;
  m=1;
  k=1;
  for (y=0; y<4; y++)
  {
    matriz_char[s][m]=num[y][k];
    s++;
  }


  s=0;
  m=2;
  k=2;
  for (y=0; y<4; y++)
  {
    matriz_char[s][m]=num[y][k];
    s++;
  }


  s=0;
  m=3;
  k=3;
  for (y=0; y<4; y++)
  {
    matriz_char[s][m]=num[y][k];
    s++;
  }



}
```

```
int matriz_or_exclusivainvuno(int HexParaDesencriptar[],int numero0,int numerito[][4],int numero1,int transfkey[][4],int numero2)
{
   int i,y,k,s,a,b;
  /* Suma en or_exclusiva */

        a=0;
        b=0;
        k=0;
        s=0;
        i=0;
        for (y=0; y<4; y++)
        {
            HexParaDesencriptar[s]=HexParaDesencriptar[i]^transfkey[y][k];
            numerito[a][b]=HexParaDesencriptar[s];
            a++;
            s++;
            i++;
        }
        k=1;
        b=1;
        a=0;
        for (y=0; y<4; y++)
        {
            HexParaDesencriptar[s]=HexParaDesencriptar[i]^transfkey[y][k];
            numerito[a][b]=HexParaDesencriptar[s];
            a++;
            s++;
            i++;
        }
        k=2;
        b=2;
        a=0;
        for (y=0; y<4; y++)
```

```
                {
                    HexParaDesencriptar[s]=HexParaDesencriptar[i]^transfkey[y][k];
                    numerito[a][b]=HexParaDesencriptar[s];
                    a++;
                    s++;
                    i++;
                }
                k=3;
                b=3;
                a=0;
                for (y=0; y<4; y++)
                {
                    HexParaDesencriptar[s]=HexParaDesencriptar[i]^transfkey[y][k];
                    numerito[a][b]=HexParaDesencriptar[s];
                    a++;
                    s++;
                    i++;
                }



            }
            int matriz_or_exclusivainv(int numerito[][4],int numero1,int transfkey[][4],int
numero2)
            {
              int i,j,y,k,s,m;
              /* Suma en or_exclusiva */


                    m=0;
                    j=0;
                    k=0;
                    s=0;
                    i=0;
                    for (y=0; y<4; y++)
                    {
                        numerito[s][m]=numerito[i][j]^transfkey[y][k];
```

```
    s++;
    i++;
}


m=1;
j=1;
k=1;
s=0;
i=0;
for (y=0; y<4; y++)
{
    numerito[s][m]=numerito[i][j]^transfkey[y][k];

    s++;
    i++;
}


m=2;
j=2;
k=2;
s=0;
i=0;
for (y=0; y<4; y++)
{
    numerito[s][m]=numerito[i][j]^transfkey[y][k];

    s++;
    i++;
}

m=3;
j=3;
k=3;
s=0;
i=0;
```

```
        for (y=0; y<4; y++)
        {
            numerito[s][m]=numerito[i][j]^transfkey[y][k];

            s++;
            i++;
        }
}

 int shift_rowinv(int row[][4],int numero2)
{
    int i,j,s,m;
    int matrizreemplazo[4][4];
    int primero;
    int primero1;
    int segundo1;
    int primero2;
    int segundo2;
    int tercero2;

    i=0;
    for (j=0; j<4; j++)

   primero2=row[1][0];
   segundo2=row[1][1];
   tercero2=row[1][2];
    i=1;
    s=1;
    m=0;
    for (j=3; j<4; j++)
    {
        matrizreemplazo[s][m]=row[i][j];
        m++;
     }

    matrizreemplazo[1][1]=primero2;
    matrizreemplazo[1][2]=segundo2;
```

```
matrizreemplazo[1][3]=tercero2;

for (m=0; m<4; m++)


primero1=row[2][0];
segundo1=row[2][1];
i=2;
s=2;
m=0;
for (j=2; j<4; j++)
{
   matrizreemplazo[s][m]=row[i][j];
   m++;
}

matrizreemplazo[2][2]=primero1;
matrizreemplazo[2][3]=segundo1;

for (m=0; m<4; m++)

primero=row[3][0];
i=3;
s=3;
m=0;
for (j=1; j<4; j++)
{
   matrizreemplazo[s][m]=row[i][j];
   m++;
}

matrizreemplazo[3][3]=primero;

for (m=0; m<4; m++)

i=0;
for (j=0; j<4; j++)
```

```
        i=1;
        s=1;
        j=0;
        for (m=0; m<4; m++)
        {
          row[i][j]=matrizreemplazo[s][m];
          j++;
        }
        i=2;
        s=2;
        j=0;
        for (m=0; m<4; m++)
        {
           row[i][j]=matrizreemplazo[s][m];
            j++;
        }
        i=3;
        s=3;
        j=0;
        for (m=0; m<4; m++)
        {
           row[i][j]=matrizreemplazo[s][m];
           j++;
        }

}

 int sub_byteinv(int sboxtransfinv[],int num1,int sub[][4],int numero1)
  {
     /* Armado de la matriz S_BOX para la transformacion SubBytes */
    int i,j,s,m;
    int auxiliar[4][4];

    /* La llave a usar sera 000102030405060708090a0b0c0d0e0f */

/* Armado de la matriz sub_byte  */
         i=0;
```

```
s=0;
m=0;
for (j=0; j<4; j++)
{
   auxiliar[s][m]=sboxtransfinv[sub[i][j]];
   m++;
}

i=1;
s=1;
m=0;
for (j=0; j<4; j++)
{
   auxiliar[s][m]=sboxtransfinv[sub[i][j]];
   m++;
}
i=2;
s=2;
m=0;
for (j=0; j<4; j++)
{
   auxiliar[s][m]=sboxtransfinv[sub[i][j]];
   m++;
}
i=3;
s=3;
m=0;
for (j=0; j<4; j++)
{
   auxiliar[s][m]=sboxtransfinv[sub[i][j]];
   m++;
}

i=0;
s=0;
j=0;
for (m=0; m<4; m++)
```

```
        {
           sub[i][j]=auxiliar[s][m];
           j++;
        }
        i=1;
        s=1;
        j=0;
        for (m=0; m<4; m++)
        {
           sub[i][j]=auxiliar[s][m];
           j++;
        }
        i=2;
        s=2;
        j=0;
        for (m=0; m<4; m++)
        {
           sub[i][j]=auxiliar[s][m];
           j++;
        }
        i=3;
        s=3;
        j=0;
        for (m=0; m<4; m++)
        {
           sub[i][j]=auxiliar[s][m];
           j++;
        }

  }
 int mix_columnsinv(int mix[][4],int numerox)
{
   int matriztransf[4][4];
   int i,j,y,k,s,m;
   int suma;

   int mixta[4][4];
```

```
 int matrizmixta[4][4];

 matriztransf[0][0]=0x0e;
 matriztransf[0][1]=0x0b;
 matriztransf[0][2]=0x0d;
 matriztransf[0][3]=0x09;
 matriztransf[1][0]=0x09;
 matriztransf[1][1]=0x0e;
 matriztransf[1][2]=0x0b;
 matriztransf[1][3]=0x0d;
 matriztransf[2][0]=0x0d;
 matriztransf[2][1]=0x09;
 matriztransf[2][2]=0x0e;
 matriztransf[2][3]=0x0b;
 matriztransf[3][0]=0x0b;
 matriztransf[3][1]=0x0d;
 matriztransf[3][2]=0x09;
 matriztransf[3][3]=0x0e;

y=0;
m=0;
for (j=0; j<4; j++)
{
 k=0;
 s=0;
 for (i=0; i<4; i++)
 {

    if (matriztransf[y][k]==0x0e)
    {
      mixta[s][m]=(mix[i][j]*0x08)^(mix[i][j]*0x04)^(mix[i][j]*0x02);
      if ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
        mixta[s][m]=mixta[s][m]^0x11b;
      else if ((mixta[s][m]>0x1ff) && (mixta[s][m]<0x3ff))
       {
         mixta[s][m]=mixta[s][m]^0x236;
         if (mixta[s][m]>0xff)
```

```
        mixta[s][m]=mixta[s][m]^0x11b;
      else
         mixta[s][m]=mixta[s][m];
    }
    else if (mixta[s][m]>0x3ff)
     {
      mixta[s][m]=mixta[s][m]^0x46c;
      if (mixta[s][m]>0x1ff)
      {
         mixta[s][m]=mixta[s][m]^0x236;
         if  (mixta[s][m]>0xff)
           mixta[s][m]=mixta[s][m]^0x11b;
         else
           mixta[s][m]=mixta[s][m];
       }

      else if  ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
        mixta[s][m]=mixta[s][m]^0x11b;
      else
        mixta[s][m]= mixta[s][m];
      }
    else
         mixta[s][m]=mixta[s][m];



   k++;
   s++;

}
else if  (matriztransf[y][k]==0x0b)
{
   mixta[s][m]=(mix[i][j]*0x08)^(mix[i][j]*0x02)^(mix[i][j]*0x01);
   if ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
     mixta[s][m]=mixta[s][m]^0x11b;
   else if ((mixta[s][m]>0x1ff) && (mixta[s][m]<0x3ff))
```

```
    {
      mixta[s][m]=mixta[s][m]^0x236;
      if (mixta[s][m]>0xff)

        mixta[s][m]=mixta[s][m]^0x11b;
      else
         mixta[s][m]=mixta[s][m];
    }
    else if (mixta[s][m]>0x3ff)
     {
      mixta[s][m]=mixta[s][m]^0x46c;
      if (mixta[s][m]>0x1ff)
      {
         mixta[s][m]=mixta[s][m]^0x236;
         if  (mixta[s][m]>0xff)
           mixta[s][m]=mixta[s][m]^0x11b;
         else
           mixta[s][m]=mixta[s][m];
       }

      else if  ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
        mixta[s][m]=mixta[s][m]^0x11b;
      else
        mixta[s][m]= mixta[s][m];
      }
    else
         mixta[s][m]=mixta[s][m];




    k++;
    s++;

}
 else if  (matriztransf[y][k]==0x0d)
  {
```

```
mixta[s][m]=(mix[i][j]*0x08)^(mix[i][j]*0x04)^(mix[i][j]*0x01);
 if ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
   mixta[s][m]=mixta[s][m]^0x11b;
else if ((mixta[s][m]>0x1ff) && (mixta[s][m]<0x3ff))
 {
   mixta[s][m]=mixta[s][m]^0x236;
   if (mixta[s][m]>0xff)

     mixta[s][m]=mixta[s][m]^0x11b;
   else
      mixta[s][m]=mixta[s][m];
 }
 else if (mixta[s][m]>0x3ff)
  {
   mixta[s][m]=mixta[s][m]^0x46c;
   if (mixta[s][m]>0x1ff)
   {
      mixta[s][m]=mixta[s][m]^0x236;
      if  (mixta[s][m]>0xff)
        mixta[s][m]=mixta[s][m]^0x11b;
      else
        mixta[s][m]=mixta[s][m];
    }

   else if  ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
     mixta[s][m]=mixta[s][m]^0x11b;
   else
     mixta[s][m]= mixta[s][m];
  }
 else
      mixta[s][m]=mixta[s][m];



k++;
s++;
```

```
}
else if  (matriztransf[y][k]==0x09)
 {
   mixta[s][m]=(mix[i][j]*0x08)^(mix[i][j]*0x01);
   if ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
     mixta[s][m]=mixta[s][m]^0x11b;
   else if ((mixta[s][m]>0x1ff) && (mixta[s][m]<0x3ff))
    {
      mixta[s][m]=mixta[s][m]^0x236;
      if (mixta[s][m]>0xff)

        mixta[s][m]=mixta[s][m]^0x11b;
      else
         mixta[s][m]=mixta[s][m];
    }
    else if (mixta[s][m]>0x3ff)
     {
      mixta[s][m]=mixta[s][m]^0x46c;
      if (mixta[s][m]>0x1ff)
      {
         mixta[s][m]=mixta[s][m]^0x236;
         if  (mixta[s][m]>0xff)
           mixta[s][m]=mixta[s][m]^0x11b;
         else
           mixta[s][m]=mixta[s][m];
       }

      else if  ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
        mixta[s][m]=mixta[s][m]^0x11b;
      else
        mixta[s][m]= mixta[s][m];
      }
    else
         mixta[s][m]=mixta[s][m];
```

```
        k++;
        s++;

     }
   }


   suma=mixta[0][m];
   for (s=1; s<4; s++)
    suma=suma^mixta[s][m];
    matrizmixta[m][y]=suma;

    m++;
}


y=1;
m=0;
for (j=0; j<4; j++)
{
  k=0;
  s=0;
  for (i=0; i<4; i++)
  {

     if (matriztransf[y][k]==0x0e)
     {
       mixta[s][m]=(mix[i][j]*0x08)^(mix[i][j]*0x04)^(mix[i][j]*0x02);
       if ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
          mixta[s][m]=mixta[s][m]^0x11b;
       else if ((mixta[s][m]>0x1ff) && (mixta[s][m]<0x3ff))
        {
          mixta[s][m]=mixta[s][m]^0x236;
          if (mixta[s][m]>0xff)

             mixta[s][m]=mixta[s][m]^0x11b;
          else
```

```
          mixta[s][m]=mixta[s][m];
    }
    else if (mixta[s][m]>0x3ff)
     {
      mixta[s][m]=mixta[s][m]^0x46c;
      if (mixta[s][m]>0x1ff)
      {
         mixta[s][m]=mixta[s][m]^0x236;
         if  (mixta[s][m]>0xff)
           mixta[s][m]=mixta[s][m]^0x11b;
         else
           mixta[s][m]=mixta[s][m];
       }

      else if  ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
        mixta[s][m]=mixta[s][m]^0x11b;
      else
        mixta[s][m]= mixta[s][m];
     }
    else
         mixta[s][m]=mixta[s][m];



   k++;
   s++;

}
else if  (matriztransf[y][k]==0x0b)
{
   mixta[s][m]=(mix[i][j]*0x08)^(mix[i][j]*0x02)^(mix[i][j]*0x01);
   if ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
     mixta[s][m]=mixta[s][m]^0x11b;
   else if ((mixta[s][m]>0x1ff) && (mixta[s][m]<0x3ff))
    {
      mixta[s][m]=mixta[s][m]^0x236;
      if (mixta[s][m]>0xff)
```

```
            mixta[s][m]=mixta[s][m]^0x11b;
          else
             mixta[s][m]=mixta[s][m];
        }
        else if (mixta[s][m]>0x3ff)
         {
          mixta[s][m]=mixta[s][m]^0x46c;
          if (mixta[s][m]>0x1ff)
          {
             mixta[s][m]=mixta[s][m]^0x236;
             if  (mixta[s][m]>0xff)
               mixta[s][m]=mixta[s][m]^0x11b;
             else
               mixta[s][m]=mixta[s][m];
           }

          else if  ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
            mixta[s][m]=mixta[s][m]^0x11b;
          else
            mixta[s][m]= mixta[s][m];
          }
        else
             mixta[s][m]=mixta[s][m];




       k++;
       s++;

}
 else if  (matriztransf[y][k]==0x0d)
  {
    mixta[s][m]=(mix[i][j]*0x08)^(mix[i][j]*0x04)^(mix[i][j]*0x01);
     if ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
       mixta[s][m]=mixta[s][m]^0x11b;
```

```
   else if ((mixta[s][m]>0x1ff) && (mixta[s][m]<0x3ff))
    {
      mixta[s][m]=mixta[s][m]^0x236;
      if (mixta[s][m]>0xff)

        mixta[s][m]=mixta[s][m]^0x11b;
      else
         mixta[s][m]=mixta[s][m];
    }
    else if (mixta[s][m]>0x3ff)
     {
      mixta[s][m]=mixta[s][m]^0x46c;
      if (mixta[s][m]>0x1ff)
      {
         mixta[s][m]=mixta[s][m]^0x236;
         if  (mixta[s][m]>0xff)
           mixta[s][m]=mixta[s][m]^0x11b;
         else
           mixta[s][m]=mixta[s][m];
       }

      else if  ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
        mixta[s][m]=mixta[s][m]^0x11b;
      else
        mixta[s][m]= mixta[s][m];
      }
    else
         mixta[s][m]=mixta[s][m];



   k++;
   s++;

}
else if  (matriztransf[y][k]==0x09)
 {
```

```
mixta[s][m]=(mix[i][j]*0x08)^(mix[i][j]*0x01);
if ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
  mixta[s][m]=mixta[s][m]^0x11b;
else if ((mixta[s][m]>0x1ff) && (mixta[s][m]<0x3ff))
 {
   mixta[s][m]=mixta[s][m]^0x236;
   if (mixta[s][m]>0xff)

     mixta[s][m]=mixta[s][m]^0x11b;
   else
      mixta[s][m]=mixta[s][m];
 }
 else if (mixta[s][m]>0x3ff)
  {
   mixta[s][m]=mixta[s][m]^0x46c;
   if (mixta[s][m]>0x1ff)
   {
      mixta[s][m]=mixta[s][m]^0x236;
      if  (mixta[s][m]>0xff)
        mixta[s][m]=mixta[s][m]^0x11b;
      else
        mixta[s][m]=mixta[s][m];
    }

   else if  ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
     mixta[s][m]=mixta[s][m]^0x11b;
   else
     mixta[s][m]= mixta[s][m];
  }
 else
      mixta[s][m]=mixta[s][m];



k++;
s++;
```

```
    }
   }


   suma=mixta[0][m];
   for (s=1; s<4; s++)
    suma=suma^mixta[s][m];
    matrizmixta[m][y]=suma;

    m++;
}

y=2;
m=0;
for (j=0; j<4; j++)
{
  k=0;
  s=0;
  for (i=0; i<4; i++)
  {

     if (matriztransf[y][k]==0x0e)
     {
       mixta[s][m]=(mix[i][j]*0x08)^(mix[i][j]*0x04)^(mix[i][j]*0x02);
       if ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
          mixta[s][m]=mixta[s][m]^0x11b;
       else if ((mixta[s][m]>0x1ff) && (mixta[s][m]<0x3ff))
        {
          mixta[s][m]=mixta[s][m]^0x236;
          if (mixta[s][m]>0xff)

             mixta[s][m]=mixta[s][m]^0x11b;
          else
              mixta[s][m]=mixta[s][m];
        }
        else if (mixta[s][m]>0x3ff)
         {
```

```
      mixta[s][m]=mixta[s][m]^0x46c;
      if (mixta[s][m]>0x1ff)
      {
         mixta[s][m]=mixta[s][m]^0x236;
         if  (mixta[s][m]>0xff)
           mixta[s][m]=mixta[s][m]^0x11b;
         else
           mixta[s][m]=mixta[s][m];
       }

      else if  ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
        mixta[s][m]=mixta[s][m]^0x11b;
      else
        mixta[s][m]= mixta[s][m];
     }
    else
         mixta[s][m]=mixta[s][m];



   k++;
   s++;

}
else if  (matriztransf[y][k]==0x0b)
{
   mixta[s][m]=(mix[i][j]*0x08)^(mix[i][j]*0x02)^(mix[i][j]*0x01);
   if ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
     mixta[s][m]=mixta[s][m]^0x11b;
   else if ((mixta[s][m]>0x1ff) && (mixta[s][m]<0x3ff))
    {
      mixta[s][m]=mixta[s][m]^0x236;
      if (mixta[s][m]>0xff)

        mixta[s][m]=mixta[s][m]^0x11b;
      else
         mixta[s][m]=mixta[s][m];
```

```
      }
      else if (mixta[s][m]>0x3ff)
       {
        mixta[s][m]=mixta[s][m]^0x46c;
        if (mixta[s][m]>0x1ff)
        {
           mixta[s][m]=mixta[s][m]^0x236;
           if  (mixta[s][m]>0xff)
             mixta[s][m]=mixta[s][m]^0x11b;
           else
             mixta[s][m]=mixta[s][m];
         }

        else if  ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
          mixta[s][m]=mixta[s][m]^0x11b;
        else
          mixta[s][m]= mixta[s][m];
       }
      else
           mixta[s][m]=mixta[s][m];




     k++;
     s++;

}
 else if  (matriztransf[y][k]==0x0d)
  {
     mixta[s][m]=(mix[i][j]*0x08)^(mix[i][j]*0x04)^(mix[i][j]*0x01);
      if ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
        mixta[s][m]=mixta[s][m]^0x11b;
     else if ((mixta[s][m]>0x1ff) && (mixta[s][m]<0x3ff))
      {
        mixta[s][m]=mixta[s][m]^0x236;
        if (mixta[s][m]>0xff)
```

```
          mixta[s][m]=mixta[s][m]^0x11b;
        else
           mixta[s][m]=mixta[s][m];
      }
      else if (mixta[s][m]>0x3ff)
       {
        mixta[s][m]=mixta[s][m]^0x46c;
        if (mixta[s][m]>0x1ff)
        {
           mixta[s][m]=mixta[s][m]^0x236;
           if  (mixta[s][m]>0xff)
             mixta[s][m]=mixta[s][m]^0x11b;
           else
             mixta[s][m]=mixta[s][m];
         }

        else if  ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
          mixta[s][m]=mixta[s][m]^0x11b;
        else
          mixta[s][m]= mixta[s][m];
        }
      else
           mixta[s][m]=mixta[s][m];



     k++;
     s++;

}
else if  (matriztransf[y][k]==0x09)
 {
    mixta[s][m]=(mix[i][j]*0x08)^(mix[i][j]*0x01);
    if ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
      mixta[s][m]=mixta[s][m]^0x11b;
    else if ((mixta[s][m]>0x1ff) && (mixta[s][m]<0x3ff))
```

```
        {
          mixta[s][m]=mixta[s][m]^0x236;
          if (mixta[s][m]>0xff)

            mixta[s][m]=mixta[s][m]^0x11b;
          else
             mixta[s][m]=mixta[s][m];
        }
        else if (mixta[s][m]>0x3ff)
         {
          mixta[s][m]=mixta[s][m]^0x46c;
          if (mixta[s][m]>0x1ff)
          {
             mixta[s][m]=mixta[s][m]^0x236;
             if  (mixta[s][m]>0xff)
               mixta[s][m]=mixta[s][m]^0x11b;
             else
               mixta[s][m]=mixta[s][m];
           }

          else if  ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
            mixta[s][m]=mixta[s][m]^0x11b;
          else
            mixta[s][m]= mixta[s][m];
          }
        else
             mixta[s][m]=mixta[s][m];



       k++;
       s++;

   }
}
```

```
   suma=mixta[0][m];
   for (s=1; s<4; s++)
    suma=suma^mixta[s][m];
   matrizmixta[m][y]=suma;

   m++;
}

y=3;
m=0;
for (j=0; j<4; j++)
{
 k=0;
 s=0;
 for (i=0; i<4; i++)
 {

    if (matriztransf[y][k]==0x0e)
    {
      mixta[s][m]=(mix[i][j]*0x08)^(mix[i][j]*0x04)^(mix[i][j]*0x02);
      if ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
        mixta[s][m]=mixta[s][m]^0x11b;
      else if ((mixta[s][m]>0x1ff) && (mixta[s][m]<0x3ff))
       {
         mixta[s][m]=mixta[s][m]^0x236;
         if (mixta[s][m]>0xff)

            mixta[s][m]=mixta[s][m]^0x11b;
         else
            mixta[s][m]=mixta[s][m];
       }
       else if (mixta[s][m]>0x3ff)
        {
         mixta[s][m]=mixta[s][m]^0x46c;
         if (mixta[s][m]>0x1ff)
         {
            mixta[s][m]=mixta[s][m]^0x236;
```

```
        if  (mixta[s][m]>0xff)
          mixta[s][m]=mixta[s][m]^0x11b;
        else
          mixta[s][m]=mixta[s][m];
       }

      else if  ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
        mixta[s][m]=mixta[s][m]^0x11b;
      else
        mixta[s][m]= mixta[s][m];
     }
    else
         mixta[s][m]=mixta[s][m];



  k++;
  s++;

}
else if  (matriztransf[y][k]==0x0b)
{
  mixta[s][m]=(mix[i][j]*0x08)^(mix[i][j]*0x02)^(mix[i][j]*0x01);
  if ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
    mixta[s][m]=mixta[s][m]^0x11b;
  else if ((mixta[s][m]>0x1ff) && (mixta[s][m]<0x3ff))
   {
    mixta[s][m]=mixta[s][m]^0x236;
    if (mixta[s][m]>0xff)

      mixta[s][m]=mixta[s][m]^0x11b;
    else
       mixta[s][m]=mixta[s][m];
   }
   else if (mixta[s][m]>0x3ff)
    {
    mixta[s][m]=mixta[s][m]^0x46c;
```

```
        if (mixta[s][m]>0x1ff)
        {
           mixta[s][m]=mixta[s][m]^0x236;
           if  (mixta[s][m]>0xff)
             mixta[s][m]=mixta[s][m]^0x11b;
           else
             mixta[s][m]=mixta[s][m];
         }

        else if  ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
          mixta[s][m]=mixta[s][m]^0x11b;
        else
          mixta[s][m]= mixta[s][m];
       }
      else
           mixta[s][m]=mixta[s][m];




    k++;
    s++;

}
 else if  (matriztransf[y][k]==0x0d)
  {
    mixta[s][m]=(mix[i][j]*0x08)^(mix[i][j]*0x04)^(mix[i][j]*0x01);
     if ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
       mixta[s][m]=mixta[s][m]^0x11b;
    else if ((mixta[s][m]>0x1ff) && (mixta[s][m]<0x3ff))
     {
       mixta[s][m]=mixta[s][m]^0x236;
       if (mixta[s][m]>0xff)

         mixta[s][m]=mixta[s][m]^0x11b;
       else
          mixta[s][m]=mixta[s][m];
```

```
    }
    else if (mixta[s][m]>0x3ff)
     {
      mixta[s][m]=mixta[s][m]^0x46c;
      if (mixta[s][m]>0x1ff)
      {
         mixta[s][m]=mixta[s][m]^0x236;
         if  (mixta[s][m]>0xff)
           mixta[s][m]=mixta[s][m]^0x11b;
         else
           mixta[s][m]=mixta[s][m];
       }

      else if  ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
        mixta[s][m]=mixta[s][m]^0x11b;
      else
        mixta[s][m]= mixta[s][m];
      }
    else
         mixta[s][m]=mixta[s][m];



   k++;
   s++;

}
else if  (matriztransf[y][k]==0x09)
 {
   mixta[s][m]=(mix[i][j]*0x08)^(mix[i][j]*0x01);
   if ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
     mixta[s][m]=mixta[s][m]^0x11b;
   else if ((mixta[s][m]>0x1ff) && (mixta[s][m]<0x3ff))
    {
      mixta[s][m]=mixta[s][m]^0x236;
      if (mixta[s][m]>0xff)
```

```
            mixta[s][m]=mixta[s][m]^0x11b;
          else
             mixta[s][m]=mixta[s][m];
        }
        else if (mixta[s][m]>0x3ff)
         {
          mixta[s][m]=mixta[s][m]^0x46c;
          if (mixta[s][m]>0x1ff)
          {
             mixta[s][m]=mixta[s][m]^0x236;
             if  (mixta[s][m]>0xff)
               mixta[s][m]=mixta[s][m]^0x11b;
             else
               mixta[s][m]=mixta[s][m];
           }

          else if  ((mixta[s][m]>0xff) && (mixta[s][m]<0x1ff))
            mixta[s][m]=mixta[s][m]^0x11b;
          else
            mixta[s][m]= mixta[s][m];
          }
        else
             mixta[s][m]=mixta[s][m];



       k++;
      s++;

   }
}


suma=mixta[0][m];
for (s=1; s<4; s++)
  suma=suma^mixta[s][m];
 matrizmixta[m][y]=suma;
```

```
   m++;
}

 i=0;
    y=0;
    m=0;
    for(j=0; j<4; j++)
 {
            mix[i][j]=matrizmixta[m][y];
   m++;
    }

i=1;
    y=1;
    m=0;
    for(j=0; j<4; j++)
 {
            mix[i][j]=matrizmixta[m][y];
   m++;
    }

i=2;
    y=2;
    m=0;
    for(j=0; j<4; j++)
 {
            mix[i][j]=matrizmixta[m][y];
            m++;
    }


i=3;
    y=3;
    m=0;
    for(j=0; j<4; j++)
 {
```

```
                mix[i][j]=matrizmixta[m][y];
                m++;
        }
}

int exp_shiftrow_llaveinv(int llavecita[][4],int numeros1)
{
    int i,j;
    int primero;

    primero=llavecita[0][3];
      j=3;
      for (i=0; i<3; i++)
           llavecita[i][j]=llavecita[i+1][j];
      llavecita[3][3]=primero;



}
 int exp_sub_byte_llaveinv(int sboxtransf[],int num1,int sub[][4],int numero1)
 {
  int y,k,i,j;
  int auxiliar[4][4];

     k=3;
     j=3;
     i=0;
     for (y=0; y<4; y++)
     {
          auxiliar[i][j]=sboxtransf[sub[y][k]];
          i++;
     }

     k=3;
     j=3;
     i=0;
     for (y=0; y<4; y++)
     {
```

```
            sub[y][k]=auxiliar[i][j];
            i++;
          }


        }
        int exp_or_exc1_llaveinv(int ciclo1,int vector1[],int num1,int llavecita[][4],int
numero1)
        {

          int y,i,j;
          int valor;
          int resultado;

          if (ciclo1==0)
            vector1[0]=vector1[0];
          else
          {
            if (ciclo1==8)
              vector1[0]=0x1b;
            else if (ciclo1==9)
              vector1[0]=0x36;
            else
            {
              resultado=vector1[0]*2;
              vector1[0]=resultado;
            }
          }
          j=3;
          y=0;
          for (i=0; i<4; i++)
          {
            llavecita[i][j]=llavecita[i][j]^vector1[y];
            y++;
          }
          valor=vector1[0];
```

```
}
 int exp_or_exc2_llaveinv(int reserva[][4],int num1,int llavecita[][4],int numero1)
 {
   int i,j,y,k;

   j=3;
   k=0;
   y=0;
   for (i=0; i<4; i++)
   {
    llavecita[i][j]=llavecita[i][j]^reserva[y][k];
    reserva[y][k]=llavecita[i][j];
    y++;
   }

   for (y=0; y<4; y++)


   k=1;
   y=0;
   for (i=0; i<4; i++)
   {
    llavecita[i][j]=llavecita[i][j]^reserva[y][k];
    reserva[y][k]=llavecita[i][j];
    y++;
   }


   for (y=0; y<4; y++)


   k=2;
   y=0;
   for (i=0; i<4; i++)
   {
    llavecita[i][j]=llavecita[i][j]^reserva[y][k];
    reserva[y][k]=llavecita[i][j];
```

```
      y++;
     }


     for (y=0; y<4; y++)


     k=3;
     y=0;
     for (i=0; i<4; i++)
     {
      llavecita[i][j]=llavecita[i][j]^reserva[y][k];
      reserva[y][k]=llavecita[i][j];
      y++;
     }


   }

  /*Funciones del programa de encriptado */
   int matriz_ascii(unsigned char vectorparaencriptar[],int numero2, int num[][4],int numero1)
   {
     /* Armado de la matriz en codigo ascii */
      int y,k;
     /* unsigned char pepe[16]="Hola como están!";*/
      int j=0;


      for (k=0; k<4; k++)
       for (y=0; y<numero1; y++)
       {
          num[y][k]=vectorparaencriptar[j];
          j++;
       }


   }
```

```
int matriz_or_exclusiva(int numerito[][4],int numero1,int transfkey[][4],int numero2)
{

   int i,j,y,k,p,q,s,m;




/* Suma en or_exclusiva */

        m=0;
        j=0;
        k=0;
        s=0;
        i=0;
        for (y=0; y<4; y++)
        {
            numerito[s][m]=numerito[i][j]^transfkey[y][k];

            s++;
            i++;
        }


        m=1;
        j=1;
        k=1;
        s=0;
        i=0;
        for (y=0; y<4; y++)
        {
            numerito[s][m]=numerito[i][j]^transfkey[y][k];

            s++;
            i++;
```

```
        }


        m=2;
        j=2;
        k=2;
        s=0;
        i=0;
        for (y=0; y<4; y++)
        {
            numerito[s][m]=numerito[i][j]^transfkey[y][k];

            s++;
            i++;
        }

        m=3;
        j=3;
        k=3;
        s=0;
        i=0;
        for (y=0; y<4; y++)
        {
            numerito[s][m]=numerito[i][j]^transfkey[y][k];

            s++;
            i++;
        }


}
```

```
int sub_byte(int sboxtransf[],int num1,int sub[][4],int numero1)
```

```
{
   /* Armado de la matriz S_BOX para la transformacion SubBytes */
   int i,j,y,k,s,m;
   int auxiliar[4][4];

   /* La llave a usar sera 000102030405060708090a0b0c0d0e0f */

   /* Armado de la matriz sub_byte  */



      i=0;
      s=0;
      m=0;
      for (j=0; j<4; j++)
      {
         auxiliar[s][m]=sboxtransf[sub[i][j]];
         m++;
      }

      i=1;
      s=1;
      m=0;
      for (j=0; j<4; j++)
      {
         auxiliar[s][m]=sboxtransf[sub[i][j]];
         m++;
      }
      i=2;
      s=2;
      m=0;
      for (j=0; j<4; j++)
      {
         auxiliar[s][m]=sboxtransf[sub[i][j]];
         m++;
      }
      i=3;
```

```
s=3;
m=0;
for (j=0; j<4; j++)
{
   auxiliar[s][m]=sboxtransf[sub[i][j]];
   m++;
}

i=0;
s=0;
j=0;
for (m=0; m<4; m++)
{
   sub[i][j]=auxiliar[s][m];
   j++;
}
i=1;
s=1;
j=0;
for (m=0; m<4; m++)
{
   sub[i][j]=auxiliar[s][m];
   j++;
}
i=2;
s=2;
j=0;
for (m=0; m<4; m++)
{
   sub[i][j]=auxiliar[s][m];
   j++;
}
i=3;
s=3;
j=0;
for (m=0; m<4; m++)
{
```

```
            sub[i][j]=auxiliar[s][m];
            j++;
          }
}

 int shift_row(int row[][4],int numero2)
 {
      int i,j,s,m;
      int matrizreemplazo[4][4];
      int primero;
      int primero1;
      int segundo1;
      int primero2;
      int segundo2;
      int tercero2;

      primero=row[1][0];
      i=1;
      s=1;
      m=0;
      for (j=1; j<4; j++)
      {
          matrizreemplazo[s][m]=row[i][j];
          m++;
       }

      matrizreemplazo[1][3]=primero;

      for (m=0; m<4; m++)


     primero1=row[2][0];
     segundo1=row[2][1];
      i=2;
      s=2;
      m=0;
      for (j=2; j<4; j++)
```

```
{
    matrizreemplazo[s][m]=row[i][j];
    m++;
 }

matrizreemplazo[2][2]=primero1;
matrizreemplazo[2][3]=segundo1;



primero2=row[3][0];
segundo2=row[3][1];
tercero2=row[3][2];
 i=3;
 s=3;
 m=0;
 for (j=3; j<4; j++)
 {
    matrizreemplazo[s][m]=row[i][j];
    m++;
 }

matrizreemplazo[3][1]=primero2;
matrizreemplazo[3][2]=segundo2;
matrizreemplazo[3][3]=tercero2;



  i=1;
  s=1;
  j=0;
  for (m=0; m<4; m++)
  {
     row[i][j]=matrizreemplazo[s][m];
     j++;
  }
```

```
        i=2;
        s=2;
        j=0;
        for (m=0; m<4; m++)
        {
           row[i][j]=matrizreemplazo[s][m];
           j++;
        }
        i=3;
        s=3;
        j=0;
        for (m=0; m<4; m++)
        {
           row[i][j]=matrizreemplazo[s][m];
           j++;
        }

}
 int mix_columns(int mix[][4],int numerox)
{
   int matriztransf[4][4];
   int i,j,y,k,s,m;
   int suma;


   int mixta[4][4];
   int matrizmixta[4][4];

   matriztransf[0][0]=0x02;
   matriztransf[0][1]=0x03;
   matriztransf[0][2]=0x01;
   matriztransf[0][3]=0x01;
   matriztransf[1][0]=0x01;
   matriztransf[1][1]=0x02;
   matriztransf[1][2]=0x03;
   matriztransf[1][3]=0x01;
   matriztransf[2][0]=0x01;
```

```
matriztransf[2][1]=0x01;
matriztransf[2][2]=0x02;
matriztransf[2][3]=0x03;
matriztransf[3][0]=0x03;
matriztransf[3][1]=0x01;
matriztransf[3][2]=0x01;
matriztransf[3][3]=0x02;

y=0;
m=0;
for (j=0; j<4; j++)
{
 k=0;
 s=0;
 for (i=0; i<4; i++)
 {
   if (matriztransf[y][k]==0x03)
   {
     mixta[s][m]=(mix[i][j]*0x02)^(mix[i][j]*0x01);
     if (mixta[s][m]>0xff)
        mixta[s][m]=mixta[s][m]^0x11b;
     else
        mixta[s][m]=mixta[s][m];
     k++;
     s++;

   }
   else
   {
     mixta[s][m]=(mix[i][j]*matriztransf[y][k]);
     if (mixta[s][m]>0xff)
        mixta[s][m]=mixta[s][m]^0x11b;
     else
        mixta[s][m]=mixta[s][m];
     k++;
     s++;
```

```
    }
  }


    suma=mixta[0][m];
    for (s=1; s<4; s++)
       suma=suma^mixta[s][m];
    matrizmixta[m][y]=suma;
    m++;

  }


y=1;
m=0;
for (j=0; j<4; j++)
{
 k=0;
 s=0;
 for (i=0; i<4; i++)
 {
   if (matriztransf[y][k]==0x03)
   {
     mixta[s][m]=(mix[i][j]*0x02)^(mix[i][j]*0x01);
     if (mixta[s][m]>0xff)
        mixta[s][m]=mixta[s][m]^0x11b;
     else
        mixta[s][m]=mixta[s][m];
     k++;
     s++;

   }
   else
   {
     mixta[s][m]=(mix[i][j]*matriztransf[y][k]);
     if (mixta[s][m]>0xff)
        mixta[s][m]=mixta[s][m]^0x11b;
```

```
      else
         mixta[s][m]=mixta[s][m];
      k++;
      s++;

    }
  }


    suma=mixta[0][m];
    for (s=1; s<4; s++)
       suma=suma^mixta[s][m];
    matrizmixta[m][y]=suma;
    m++;

  }


y=2;
m=0;
for (j=0; j<4; j++)
{
 k=0;
 s=0;
 for (i=0; i<4; i++)
 {
   if (matriztransf[y][k]==0x03)
   {
      mixta[s][m]=(mix[i][j]*0x02)^(mix[i][j]*0x01);
      if (mixta[s][m]>0xff)
         mixta[s][m]=mixta[s][m]^0x11b;
      else
         mixta[s][m]=mixta[s][m];
      k++;
      s++;

   }
```

```
    else
    {
      mixta[s][m]=(mix[i][j]*matriztransf[y][k]);
      if (mixta[s][m]>0xff)
         mixta[s][m]=mixta[s][m]^0x11b;
      else
         mixta[s][m]=mixta[s][m];
      k++;
      s++;

    }
  }


    suma=mixta[0][m];
    for (s=1; s<4; s++)
      suma=suma^mixta[s][m];
    matrizmixta[m][y]=suma;
    m++;

  }

y=3;
m=0;
for (j=0; j<4; j++)
{
 k=0;
 s=0;
 for (i=0; i<4; i++)
 {
   if (matriztransf[y][k]==0x03)
   {
     mixta[s][m]=(mix[i][j]*0x02)^(mix[i][j]*0x01);
     if (mixta[s][m]>0xff)
        mixta[s][m]=mixta[s][m]^0x11b;
     else
        mixta[s][m]=mixta[s][m];
```

```
      k++;
      s++;

    }
    else
    {
      mixta[s][m]=(mix[i][j]*matriztransf[y][k]);
      if (mixta[s][m]>0xff)
         mixta[s][m]=mixta[s][m]^0x11b;
      else
         mixta[s][m]=mixta[s][m];
      k++;
      s++;

    }
  }


   suma=mixta[0][m];
   for (s=1; s<4; s++)
     suma=suma^mixta[s][m];
   matrizmixta[m][y]=suma;
   m++;

}


  i=0;
  m=0;
  y=0;
  for (j=0; j<4; j++)
  {
     mix[i][j]=matrizmixta[m][y];
     m++;
  }

  i=1;
```

```
        m=0;
        y=1;
        for (j=0; j<4; j++)
        {
           mix[i][j]=matrizmixta[m][y];
           m++;
        }

        i=2;
        m=0;
        y=2;
        for (j=0; j<4; j++)
        {
           mix[i][j]=matrizmixta[m][y];
           m++;
        }

        i=3;
        m=0;
        y=3;
        for (j=0; j<4; j++)
        {
           mix[i][j]=matrizmixta[m][y];
           m++;
        }




  }
int exp_shiftrow_llave(int llavecita[][4],int numeros1)
 {
     int i,j;
     int primero;

     primero=llavecita[0][3];
```

```
        j=3;
        for (i=0; i<3; i++)
            llavecita[i][j]=llavecita[i+1][j];
        llavecita[3][3]=primero;

    }
    int exp_sub_byte_llave(int sboxtransf[],int num1,int sub[][4],int numero1)
    {
     int y,k,i,j;
     int auxiliar[4][4];

     k=3;
     j=3;
     i=0;
     for (y=0; y<4; y++)
     {
       auxiliar[i][j]=sboxtransf[sub[y][k]];
       i++;
     }

     k=3;
     j=3;
     i=0;
     for (y=0; y<4; y++)
     {
       sub[y][k]=auxiliar[i][j];
       i++;
     }


    }
    int exp_or_exc1_llave(int ciclo1,int vector1[],int num1,int llavecita[][4],int
numero1)
    {

      int y,i,j;
      int valor;
```

```
    int resultado;

    if (ciclo1==0)
      vector1[0]=vector1[0];
    else
    {
      if (ciclo1==8)
        vector1[0]=0x1b;
      else if (ciclo1==9)
        vector1[0]=0x36;
      else
      {
        resultado=vector1[0]*2;
        vector1[0]=resultado;
      }
    }
    j=3;
    y=0;
    for (i=0; i<4; i++)
    {
       llavecita[i][j]=llavecita[i][j]^vector1[y];
       y++;
    }
    valor=vector1[0];

}
 int exp_or_exc2_llave(int reserva[][4],int num1,int llavecita[][4],int numero1)
 {
   int i,j,y,k;

   j=3;
   k=0;
   y=0;
   for (i=0; i<4; i++)
   {
    llavecita[i][j]=llavecita[i][j]^reserva[y][k];
    reserva[y][k]=llavecita[i][j];
```

```
  y++;
}



k=1;
y=0;
for (i=0; i<4; i++)
{
  llavecita[i][j]=llavecita[i][j]^reserva[y][k];
  reserva[y][k]=llavecita[i][j];
  y++;
}


for (y=0; y<4; y++)

k=2;
y=0;
for (i=0; i<4; i++)
{
  llavecita[i][j]=llavecita[i][j]^reserva[y][k];
  reserva[y][k]=llavecita[i][j];
  y++;
}



k=3;
y=0;
for (i=0; i<4; i++)
{
  llavecita[i][j]=llavecita[i][j]^reserva[y][k];
  reserva[y][k]=llavecita[i][j];
  y++;
}
```

```
}


void EncriptarVector(unsigned char vectorparaencriptar[],int num0,int llave[][4],int
n01,int reserva[][4],int n02,AnsiString vectordeclave)
{
    int numeros[4][4];
    int ciclo1;
    int i,j,k,l,s,m;



      matriz_ascii(vectorparaencriptar,16,numeros,4);
      llavedeencriptado(llave,4,reserva,4,vectordeclave);
      matrices2(sboxtransf,256,vector1,4);

       ciclo1=0;
     for (l=0; l<11; l++)
     {
       if (l==0)
       {
          matriz_or_exclusiva(numeros,4,llave,4);
          sub_byte(sboxtransf,256,numeros,4);
          shift_row(numeros,4);
          mix_columns(numeros,4);
       }
       else if (l==9)
       {
          exp_shiftrow_llave(llave,4);
          exp_sub_byte_llave(sboxtransf,256,llave,4);
          exp_or_exc1_llave(ciclo1,vector1,4,llave,4);
          exp_or_exc2_llave(reserva,4,llave,4);



          matriz_or_exclusiva(numeros,4,reserva,4);
```

```
            sub_byte(sboxtransf,256,numeros,4);
            shift_row(numeros,4);
            ciclo1++;
        }
         else if (l==10)
        {
            exp_shiftrow_llave(llave,4);
            exp_sub_byte_llave(sboxtransf,256,llave,4);
            exp_or_exc1_llave(ciclo1,vector1,4,llave,4);
            exp_or_exc2_llave(reserva,4,llave,4);
            matriz_or_exclusiva(numeros,4,reserva,4);



        }
          else
        {
            exp_shiftrow_llave(llave,4);
            exp_sub_byte_llave(sboxtransf,256,llave,4);
            exp_or_exc1_llave(ciclo1,vector1,4,llave,4);
            exp_or_exc2_llave(reserva,4,llave,4);



            matriz_or_exclusiva(numeros,4,reserva,4);
            sub_byte(sboxtransf,256,numeros,4);
            shift_row(numeros,4);
            mix_columns(numeros,4);
            ciclo1++;
        }
}
  k=0;
for(j=0;j<4;j++)
  for(i=0;i<4;i++)
              {
              vectorparaencriptar[k]=numeros[i][j];
              k++;
  }
```

```
}

void DesencriptarVector(int HexParaDesencriptar[],int num1,int llave[][4],int n01,int reserva[][4], int n02,AnsiString vectordeclave)
{

int numeros[4][4];
unsigned char matriz_char[4][4];
int ciclo1;
int i,j,k,l,s,m;


 for(i=0;i<4;i++)
  for(j=0;j>4;j++)
        {
   numeros[i][j]=0;
 }

 for(i=0;i<4;i++)
  for(j=0;j>4;j++)
        {
 matriz_char[0][0]=0;
 }

 llavedeencriptado(llave,4,reserva,4,vectordeclave);
 matrices2(sboxtransf,256,vector1,4);
 matrices(sboxtransfinv,256);


 l=10;
 for (ciclo1=0; ciclo1<l; ciclo1++)
  {
   exp_shiftrow_llaveinv(llave,4);
```

```
  exp_sub_byte_llaveinv(sboxtransf,256,llave,4);
  exp_or_exc1_llaveinv(ciclo1,vector1,4,llave,4);
  exp_or_exc2_llaveinv(reserva,4,llave,4);
}

matriz_or_exclusivainvuno(HexParaDesencriptar,16,numeros,4,reserva,4);
shift_rowinv(numeros,4);
sub_byteinv(sboxtransfinv,256,numeros,4);

llavedeencriptado(llave,4,reserva,4,vectordeclave);
matrices2(sboxtransf,256,vector1,4);
matrices(sboxtransfinv,256);



l--;
for (l=9; l>0; l--)
{
   for (ciclo1=0; ciclo1<l; ciclo1++)
   {
     exp_shiftrow_llaveinv(llave,4);
     exp_sub_byte_llaveinv(sboxtransf,256,llave,4);
     exp_or_exc1_llaveinv(ciclo1,vector1,4,llave,4);
     exp_or_exc2_llaveinv(reserva,4,llave,4);
   }


   matriz_or_exclusivainv(numeros,4,reserva,4);
   mix_columnsinv(numeros,4);
   shift_rowinv(numeros,4);
   sub_byteinv(sboxtransfinv,256,numeros,4);

   llavedeencriptado(llave,4,reserva,4,vectordeclave);
   matrices2(sboxtransf,256,vector1,4);
   matrices(sboxtransfinv,256);
```

```
    }

    l=1;
    l--;

    for (ciclo1=0; ciclo1<l; ciclo1++)
        {
          exp_shiftrow_llaveinv(llave,4);
          exp_sub_byte_llaveinv(sboxtransf,256,llave,4);
          exp_or_exc1_llaveinv(ciclo1,vector1,4,llave,4);
          exp_or_exc2_llaveinv(reserva,4,llave,4);
        }
        matriz_or_exclusivainv(numeros,4,reserva,4);
        matriz_caracteres(matriz_char,4,numeros,4);

  k=0;
    for(j=0;j<4;j++)
            for(i=0;i<4;i++) {
                    HexParaDesencriptar[k]=matriz_char[i][j];
                    k++;
            }


}
```

SECCIÓN X

ANEXO B

CÓDIGO FUENTE DEL PROGRAMA CORRESPONDIENTE A LA INTERFAZ GRÁFICA

El siguiente es el código fuente correspondiente al archivo de extensión .cpp, que contiene la programación de la interfaz gráfica.

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "proyectitodeencriptado.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

   #include <iostream.h>
   #include <fnesdeencriptadoC.h>
   #include <fstream.h>
   #include <conio.h>
   #include <systdate.h>
   #include <dir.h>
   #include <dos.h>
   #include <stdio.h>
   #include <stdlib.h>
   AnsiString filename;
   AnsiString filenameelegirarchaenc;
   AnsiString filenameabrirarchadesenc;
   AnsiString filenameelegirarchivoadesencriptar;
   int numeros[4][4];
     int s,m,indice,cont,cont1,contador;
     unsigned char matriz_char[4][4];
     char cadenadesencriptada[16];
```

```
    unsigned char VectorParaDesencriptar[16];
    int HexParaDesencriptar[16],nametotaldesenc,nametotal3desenc;
    char letra;
    int letra1;
    int TamanoArchivo,TamanoArchivo1,context;
    AnsiString Temporal,Temporal1,Temporal2;
    TDateTime HoraInicio,HoraFin;
    ifstream::pos_type q,q1;
    int pepito,p,nametotal;
    div_t nametotal1,nametotal2,nametotal3;
    int ciclo1,llave[4][4],reserva[4][4];
    unsigned char vectorparaencriptar[16];
    int cont2,cont3;
    int i,j,k,l,a,b,n;
    char *vector;
    AnsiString vectordeclave;
    char *nombredearchivo;
    int flags;
    char name[MAXFILE];
    char ext[MAXEXT];
    char drive[MAXDRIVE];
    char dir[MAXDIR];
    int namecont,nameext;
    char nombremasextension[100],soloextension[16];


void __fastcall TForm1::FormShow(TObject *Sender)
{
    GBdepresentacion->Visible=true;
    GBdepresentacion->Enabled=true;
    RBencriptar->Visible=false;
    RBencriptar->Enabled=false;
    RBdesencriptar->Visible=false;
    RBdesencriptar->Enabled=false;
    GBpaneldeencrip->Visible=false;
    GBpaneldeencrip->Enabled=false;
    GBpaneldedesenc->Visible=false;
```

```
    GBpaneldedesenc->Enabled=false;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::BTNdepresentsiguienteClick(TObject *Sender)
{
 GBdepresentacion->Visible=false;
 GBdepresentacion->Enabled=false;
 RBencriptar->Visible=true;
 RBencriptar->Enabled=true;
 RBdesencriptar->Visible=true;
 RBdesencriptar->Enabled=true;
 GBpaneldeencrip->Visible=true;
 GBpaneldeencrip->Enabled=true;
 GBpaneldedesenc->Visible=false;
 GBpaneldedesenc->Enabled=false;
}
//---------------------------------------------------------------------------
 void __fastcall TForm1::BTNdepresentantdeencClick(TObject *Sender)
{
    Edeclavedeenc->Text="";
    Earchaenc->Text=" ";
    Earchenc->Text=" ";
    BTNenc->Enabled=false;
    GBdepresentacion->Visible=true;
    GBdepresentacion->Enabled=true;
    RBencriptar->Visible=false;
    RBencriptar->Enabled=false;
    RBencriptar->Checked=true;
    RBdesencriptar->Visible=false;
    RBdesencriptar->Enabled=false;
    RBdesencriptar->Checked=false;
    GBpaneldeencrip->Visible=false;
    GBpaneldeencrip->Enabled=false;
    GBpaneldedesenc->Visible=false;
    GBpaneldedesenc->Enabled=false;
}
//---------------------------------------------------------------------------
```

```
 void __fastcall TForm1::BTNdepresentantdedesencClick(TObject *Sender)
{
    Edeclavededesenc->Text="";
    Earchadesenc->Text=" ";
    Earchdesenc->Text=" ";
    BTNdesenc->Enabled=false;
    GBdepresentacion->Visible=true;
    GBdepresentacion->Enabled=true;
    RBencriptar->Visible=false;
    RBencriptar->Enabled=false;
    RBencriptar->Checked=true;
    RBdesencriptar->Visible=false;
    RBdesencriptar->Enabled=false;
    RBdesencriptar->Checked=false;
    GBpaneldeencrip->Visible=false;
    GBpaneldeencrip->Enabled=false;
    GBpaneldedesenc->Visible=false;
    GBpaneldedesenc->Enabled=false;
}
//---------------------------------------------------------------------------



void __fastcall TForm1::RBencriptarClick(TObject *Sender)
{
   Edeclavededesenc->Text="";
   Earchadesenc->Text=" ";
   Earchdesenc->Text=" ";
   BTNdesenc->Enabled=false;
   GBpaneldeencrip->Visible=true;
   GBpaneldeencrip->Enabled=true;
   GBpaneldedesenc->Visible=false;
   GBpaneldedesenc->Enabled=false;

}
//---------------------------------------------------------------------------
void __fastcall TForm1::RBdesencriptarClick(TObject *Sender)
```

```
{
    Edeclavedeenc->Text="";
    Earchaenc->Text=" ";
    Earchenc->Text=" ";
    BTNenc->Enabled=false;
    GBpaneldeencrip->Visible=false;
    GBpaneldeencrip->Enabled=false;
    GBpaneldedesenc->Visible=true;
    GBpaneldedesenc->Enabled=true;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::BTNarchaencClick(TObject *Sender)
{
   OpenDialog1->Filter=AnsiString("archivos pdf|*.pdf|documento word|*.doc|pagina web|*.htm|texto|*.txt|todos los archivos|*.*");
    if (OpenDialog1->Execute()){
        filename = OpenDialog1->FileName;

    };

    Earchaenc->Text=filename;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::EarchencMouseDown(TObject *Sender,
     TMouseButton Button, TShiftState Shift, int X, int Y)
{
   BTNenc->Enabled=true;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::EarchencKeyDown(TObject *Sender, WORD &Key,
     TShiftState Shift)
{
   BTNenc->Enabled=true;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::EarchencChange(TObject *Sender)
{
  BTNenc->Enabled=true;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::BTNarchencClick(TObject *Sender)
{
  OpenDialog1->Filter=AnsiString("archivos pdf|*.pdf|documento word|*.doc|pagina web|*.htm|texto|*.txt|todos los archivos|*.*");
   if (OpenDialog1->Execute()){
      filenameelegirarchaenc = OpenDialog1->FileName;

   };
   Earchenc->Text=filenameelegirarchaenc;
   BTNenc->Enabled=true;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::BTNarchadesencClick(TObject *Sender)
{
  OpenDialog1->Filter=AnsiString("archivos pdf|*.pdf|documento word|*.doc|pagina web|*.htm|texto|*.txt|todos los archivos|*.*");
   if (OpenDialog1->Execute()){
      filenameelegirarchivoadesencriptar = OpenDialog1->FileName;


   Earchadesenc->Text=filenameelegirarchivoadesencriptar;

  }//fin del if del open dialog
  else
  {
     Earchadesenc->Text=" ";
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::EarchdesencMouseDown(TObject *Sender,
     TMouseButton Button, TShiftState Shift, int X, int Y)
```

```
{
    BTNdesenc->Enabled=true;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::EarchdesencKeyDown(TObject *Sender, WORD &Key,
     TShiftState Shift)
{
    BTNdesenc->Enabled=true;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::EarchdesencChange(TObject *Sender)
{
    BTNdesenc->Enabled=true;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::BTNarchdesencClick(TObject *Sender)
{
   OpenDialog1->Filter=AnsiString("archivos                    pdf|*.pdf|documento
word|*.doc|pagina web|*.htm|texto|*.txt|todos los archivos|*.*");
   if (OpenDialog1->Execute()){
        filenameabrirarchadesenc= OpenDialog1->FileName;

    };
    Earchdesenc->Text=filenameabrirarchadesenc;
    BTNdesenc->Enabled=true;
}
//---------------------------------------------------------------------------




void __fastcall TForm1::BTNencClick(TObject *Sender)
{
   ProgressBar1->Visible=true;
```

```
ProgressBar1->Enabled=true;

for (m=0; m<4; m++)
        for (s=0;s<4; s++)
           llave[s][m]=0;

       for (m=0; m<4; m++)
         for (s=0;s<4; s++)
           reserva[s][m]=0;

      vectordeclave=Edeclavedeenc->Text;

filename=Earchaenc->Text;
filenameelegirarchaenc=Earchenc->Text;
char* elegir=filenameelegirarchaenc.c_str();
ofstream archivodeencriptado(elegir,ios_base::out|ios_base::binary);

  char* pepe =filename.c_str();
  ifstream Archivo(pepe,ios_base::in|ios_base::binary);
  Archivo.seekg(0,ios_base::end);
  p=Archivo.tellg();
  Temporal=p;
   for(i=0;i<16;i++){
         vectorparaencriptar[i]=0;
         }
   vector=Temporal.c_str();
   for(i=0;i<16;i++){
         if(vector[i]!='\0'){
         vectorparaencriptar[i]=vector[i];
         }
         else{
               break;
         }
   }


  EncriptarVector(vectorparaencriptar,16,llave,4,reserva,4,vectordeclave);
```

```
for (a=0;a<16;a++){
    /*Transformo el código del caracter encriptado en su caracter ASCII correspondiente*/
     letra=vectorparaencriptar[a];
     /*Guardo el caracter encriptado*/
     archivodeencriptado << letra;
}//cierre FOR a

/*Encriptado del nombre del archivo*/

  nombredearchivo=filename.c_str();
  flags=fnsplit(nombredearchivo,drive,dir,name,ext);

  for (b=0; b<100; b++)
     nombremasextension[b]=0;

  for (a=0; a<16; a++)
     vectorparaencriptar[a]=0;

  for (m=0; m<16; m++)
     soloextension[m]=0;



  namecont=0;
  b=0;
  while (name[namecont]!='\0')
  {
    nombremasextension[b]=name[namecont];
    b++;
    namecont++;
  }

  nameext=0;
```

```
m=0;
while (ext[nameext]!='\0')
{
  nombremasextension[b]=ext[nameext];
  soloextension[m]=ext[nameext];
  b++;
  nameext++;
  m++;
}
   Temporal=nameext;
   for(i=0;i<16;i++){
        vectorparaencriptar[i]=0;
        }
   vector=Temporal.c_str();
   for(i=0;i<16;i++){
        if(vector[i]!='\0'){
        vectorparaencriptar[i]=vector[i];
        }
        else{
             break;
        }
   }
  EncriptarVector(vectorparaencriptar,16,llave,4,reserva,4,vectordeclave);
  for (a=0;a<16;a++){
    /*Transformo el código del caracter encriptado en su caracter ASCII correspondiente*/
    letra=vectorparaencriptar[a];
    /*Guardo el caracter encriptado*/
    archivodeencriptado << letra;
}//cierre FOR a

    nametotal=namecont+nameext;
    Temporal=nametotal;
    for(i=0;i<16;i++){
        vectorparaencriptar[i]=0;
        }
    vector=Temporal.c_str();
```

```
            for(i=0;i<16;i++){
                if(vector[i]!='\0'){
                vectorparaencriptar[i]=vector[i];
                }
                else{
                    break;
                }
            }
          EncriptarVector(vectorparaencriptar,16,llave,4,reserva,4,vectordeclave);
          for (a=0;a<16;a++){
            /*Transformo el código del caracter encriptado en su caracter ASCII correspondiente*/
            letra=vectorparaencriptar[a];
            /*Guardo el caracter encriptado*/
            archivodeencriptado << letra;
        }//cierre FOR a

          nametotal1=div(nametotal,16);
          nametotal1.quot;
          nametotal1.rem;
          nametotal2=div(nametotal,nametotal);
          nametotal2.quot;
          nametotal2.rem;
          nametotal3.quot=nametotal1.quot+nametotal2.quot;
           Temporal=nametotal3.quot;
            for(i=0;i<16;i++){
                vectorparaencriptar[i]=0;
                }
            vector=Temporal.c_str();
            for(i=0;i<16;i++){
                if(vector[i]!='\0'){
                vectorparaencriptar[i]=vector[i];
                }
                else{
                    break;
                }
            }
```

```
            EncriptarVector(vectorparaencriptar,16,llave,4,reserva,4,vectordeclave);
            for (a=0;a<16;a++){
              /*Transformo el código del caracter encriptado en su caracter ASCII correspondiente*/
              letra=vectorparaencriptar[a];
              /*Guardo el caracter encriptado*/
              archivodeencriptado << letra;
          }//cierre FOR a

              Temporal=soloextension;
              for(i=0;i<16;i++){
                    vectorparaencriptar[i]=0;
                    }
              vector=Temporal.c_str();
              for(i=0;i<16;i++){
                    if(vector[i]!='\0'){
                    vectorparaencriptar[i]=vector[i];
                    }
                    else{
                          break;
                    }
              }
            EncriptarVector(vectorparaencriptar,16,llave,4,reserva,4,vectordeclave);

            for (a=0;a<16; a++){
              /*Transformo el código del caracter encriptado en su caracter ASCII correspondiente*/
              letra=vectorparaencriptar[a];
              /*Guardo el caracter encriptado*/
              archivodeencriptado << letra;

          }//cierre FOR a

          Archivo.seekg(0,ios_base::beg) ;//Comienzo del archivo


          /*Encriptado del archivo*/
          while (!Archivo.eof())
```

```
{
        Archivo.read(vectorparaencriptar,sizeof(vectorparaencriptar));
        EncriptarVector(vectorparaencriptar,16,llave,4,reserva,4,vectordeclave);
        q=Archivo.tellg();
        ProgressBar1->Position=q*100/p;

for (i=0;i<16;i++){
     /*Transformo el código del caracter encriptado en su caracter ASCII correspondiente*/
     letra=vectorparaencriptar[i];
     /*Guardo el caracter encriptado*/
     archivodeencriptado << letra;

     }//cierre FOR i

} //cierre while
  ProgressBar1->Visible=false;
  ProgressBar1->Enabled=false;

  ShowMessage(AnsiString("El encriptado ha finalizado"));
}//Cierre btndeencriptarClick
//---------------------------------------------------------------------------

void __fastcall TForm1::BTNdesencClick(TObject *Sender)
{
  ProgressBar1->Visible=true;
  ProgressBar1->Enabled=true;

  for (m=0; m<4; m++)
       for (s=0;s<4; s++)
          llave[s][m]=0;

       for (m=0; m<4; m++)
        for (s=0;s<4; s++)
          reserva[s][m]=0;
```

```
            vectordeclave=Edeclavededesenc->Text;

         filenameelegirarchivoadesencriptar=Earchadesenc->Text;

        //Abrir el Arch encriptado para entrada de datos
        char* elegir=filenameelegirarchivoadesencriptar.c_str();
        ifstream ArchivoADesencriptar(elegir,ios_base::in|ios_base::binary);
        //Tomo los primeros 16 caracteres, tamaño de archivo
        ArchivoADesencriptar.read(VectorParaDesencriptar,sizeof(VectorParaDesencriptar)
);
        cont1=0;
          for (cont=0; cont<16; cont++)
            {
            HexParaDesencriptar[cont1]=VectorParaDesencriptar[cont];
            cont1++;
            }
          DesencriptarVector(HexParaDesencriptar,16,llave,4,reserva,4,vectordeclave);
          Temporal="";
          for(i=0;i<16;i++){
            letra=HexParaDesencriptar[i];
            Temporal=Temporal + letra;
          }

          TamanoArchivo1=Temporal.ToInt();
          TamanoArchivo=Temporal.ToInt();
        //Tomo los primeros 16 caracteres correspondientes al tamaño de la extension del
archivo
        ArchivoADesencriptar.read(VectorParaDesencriptar,sizeof(VectorParaDesencriptar)
);
        cont1=0;
          for (cont=0; cont<16; cont++)
            {
            HexParaDesencriptar[cont1]=VectorParaDesencriptar[cont];
            cont1++;
            }
          DesencriptarVector(HexParaDesencriptar,16,llave,4,reserva,4,vectordeclave);
          Temporal="";
          for(i=0;i<16;i++){
```

```
                letra=HexParaDesencriptar[i];
                Temporal=Temporal + letra;
              }
               context=Temporal.ToInt();

        //Tomo los 16 caracteres correspondientes a nametotal
        ArchivoADesencriptar.read(VectorParaDesencriptar,sizeof(VectorParaDesencriptar)
);
        cont1=0;
                for (cont=0; cont<16; cont++)
                {
                  HexParaDesencriptar[cont1]=VectorParaDesencriptar[cont];
                  cont1++;
                }

DesencriptarVector(HexParaDesencriptar,16,llave,4,reserva,4,vectordeclave);
                Temporal="";
                 for(i=0;i<16;i++){
                    letra=HexParaDesencriptar[i];
                    Temporal=Temporal + letra;
                 }

                 nametotaldesenc=Temporal.ToInt();

        //Tomo los 16 caracteres correspondientes a nametotal3.quot
        ArchivoADesencriptar.read(VectorParaDesencriptar,sizeof(VectorParaDesencriptar)
);
        cont1=0;
                for (cont=0; cont<16; cont++)
                {
                  HexParaDesencriptar[cont1]=VectorParaDesencriptar[cont];
                  cont1++;
                }

DesencriptarVector(HexParaDesencriptar,16,llave,4,reserva,4,vectordeclave);
                 Temporal="";
                 for(i=0;i<16;i++){
                    letra=HexParaDesencriptar[i];
```

```
                    Temporal=Temporal + letra;
                  }

                  nametotal3desenc=Temporal.ToInt();

          /* Tomo los caracteres correspondientes al nombre de la extensión del archivo*/
          ArchivoADesencriptar.read(VectorParaDesencriptar,sizeof(VectorParaDesencriptar)
);
          cont1=0;
                for (cont=0; cont<16; cont++)
                {
                  HexParaDesencriptar[cont1]=VectorParaDesencriptar[cont];
                  cont1++;
                }

DesencriptarVector(HexParaDesencriptar,16,llave,4,reserva,4,vectordeclave);

                  for (m=0; m<16; m++)
                  {
                    if (context<=0)
                          break;
                    letra=HexParaDesencriptar[m];
                    Earchdesenc->Text=Earchdesenc->Text+letra;
                    context--;
                  }
                  filenameabrirarchadesenc=Earchdesenc->Text;

          //Abro el arch en el cual desencripto
          char* desen=filenameabrirarchadesenc.c_str();
          ofstream desencriptado(desen,ios_base::out|ios_base::binary);

          while (!ArchivoADesencriptar.eof())
                {

ArchivoADesencriptar.read(VectorParaDesencriptar,sizeof(VectorParaDesencriptar));

                cont1=0;
                for (cont=0; cont<16; cont++)
```

```
                {
                  HexParaDesencriptar[cont1]=VectorParaDesencriptar[cont];
                  cont1++;
                }

DesencriptarVector(HexParaDesencriptar,16,llave,4,reserva,4,vectordeclave);
                q=ArchivoADesencriptar.tellg();
                ProgressBar1->Position=q*100/TamanoArchivo1;

                 for (m=0; m<16; m++)
                  {
                    if (TamanoArchivo<=0)
                          break;
                    letra=HexParaDesencriptar[m];
                    desencriptado << letra;
                    TamanoArchivo--;
                  }




               }//while de archivo
                ProgressBar1->Visible=false;
                ProgressBar1->Enabled=false;

            ShowMessage(AnsiString("El desencriptado ha finalizado"));
            }
            //---------------------------------------------------------------------------
```