SEP

SECRETARÍA DE EDUCACIÓN PÚBLICA

INSTITUTO TECNOLÓGICO DE TUXTLA GUTIÉRREZ

REPORTE FINAL DE RECIDENCIA

LUGAR DE RESIDENCIA:

LABORATORIO DE ELECTRONICA DEL I.T.T.G.

NOMBRE DEL PROYECTO

“Arquitectura Reconfigurable Para La Implementación De Algoritmos De Criptología Y Compresión De Datos”.

ASESOR:

Dr. Héctor Ricardo Hernández de León

REVISORES:

Dr. Jorge Luis Camas Anzueto
Dr. Héctor Ricardo Hernández de León

RESIDENTES:
Juan Enrique Coutiño Ruíz No. 05270256
José Angel de la Cruz Martínez No. 05270261

Carrera: Ingeniería en Electrónica
Semestre: 10°

Febrero - Junio de 2010

RESUMEN

En esta residencia se plantea una propuesta para el diseño e implementación de algoritmos de criptografía y compresión de datos en base a una arquitectura de tipo FPGA (*Field Programmble Gate Array*). Este tipo de dispositivo se enlazara vía USB con un CPU para trabajar con el software LabView 8.6 para tener una interfaz grafica y así mismo realizar la tarea del encriptado de datos por medio del algoritmo de encriptación Rijndael, diseñado por Joan Daemen y Vincent Rijmen que fue elegido como el algoritmo de encriptación estándar (AES). El algoritmo Rijndael es un sistema simétrico de cifrado por bloques, por tanto utiliza la misma clave para el proceso de cifrado como para el proceso de descifrado. Su diseño permite la utilización de claves de sistema con longitud variable siempre que sea múltiplo de 4 bytes. La longitud de las claves utilizadas por defecto son 128 (AES 128), 192 (AES 192) y 256 (AES 256) bits. El algoritmo permite que se utilicen bloques de información con un tamaño variable siempre y cuando sea múltiplo de 4 bytes, siendo el tamaño mínimo recomendado 128 bits (16 bytes) que es el cual trabajaremos en esta residencia.

Los problemas más importantes que se plantean en una solución como la descrita en esta residencia son los relacionados con la implementación en un algoritmo AES 128 en un FPGA. El número de operaciones que se tienen que realizar para llevar a cabo este algoritmo son bastantes largas ya que el algoritmo tiene que hacer determinadas rondas que estas dependen del tamaño del algoritmo en nuestro caso es el AES 128 bits por lo tanto realizara 10 rondas, que son de tipo secuencial.

ÍNDICE

Página

CAPITULO I

1.1.- INTRODUCCIÓN.

En el mundo de información en el cual vivimos actualmente, constantemente, en cualquier actividad científico-técnica o profesional, se plantea la necesidad de transmitir o almacenar un gran número de datos, número que está creciendo de forma explosiva.

Entre las herramientas con las que hoy se cuentan son las técnicas de compresión de datos que reducen el número de bits necesarios para representar la información, lo cual genera enormes ventajas en el momento de transmitir, y además para blindar esa información contamos con la Encriptación, que nos proporciona seguridad para que personas ajenas al archivo puedan conocerlo, si de información confidencial o importante se tratase, como alguna transacción bancaria.

En los siguientes apartados se describe y muestra el diseño, junto a la implementación de un sistema, que tiene la capacidad de Comprimir y Encriptar archivos de texto plano, mediante un FPGA (*Field Programmble Gate Array por sus siglas en ingles*) y la PC. Bajo este sistema se obtiene una compresión considerable y una buena encriptación de archivos de texto.

1.2.- DEFINICIÓN DEL PROBLEMA.

El proyecto se origina a través de la creciente necesidad por comprimir bits de información, debido a que cada día contamos con más y más información, que es prescindible portarla en nuestros Dispositivos de Almacenamiento (DA), pero su capacidad está limitada o bien nuestros archivos son inmensos, con la compresión conseguimos disminuir el tamaño de los ficheros para así lograr la meta fijada, hacer pequeño algo grande y de esta manera optimizamos nuestros DA.

Pero cuando hablamos de información confidencial o de suma importancia para dependencias bancarias, gubernamentales ó de Centros de investigaciones, se debe contar con un blindaje para que esa información no sea utilizada por individuos ajenos a dicha información.

Hoy por hoy, estas instituciones invierten mucho dinero para mantener a salvo bases de datos que solo a ellos les concierne y que de caer en manos erróneas podrían causar un caos. Es aquí donde la Encriptación juega un papel importantísimo en cuanto a mantener a salvo información se trata.

La transmisión de archivos es un proceso inevitable, en la realización de esto es necesario para aquellos sistemas que lo requieren, la encriptación y compresión de archivos electrónicos. La encriptación genera seguridad, la compresión es un complemento para una mejor transmisión.

1.3.- OBJETIVOS.

1.3.1.- Objetivo General.

Diseñar e implementar un circuito digital basado en FPGA's que permitan la compresión y encriptación de archivos electrónicos.

1.3.2.- Objetivos Específicos.

- Conocer los diferentes algoritmos de compresión y encriptación de archivos electrónicos actuales y sus implementaciones en Hardware.
- Seleccionar los algoritmos de compresión y encriptación de archivos electrónicos que serán implementados en el circuito digital, basado en FPGA's.
- Diseñar la interfaz para la comunicación de la tarjeta Spartan 3E con LabView 8.6.
- Utilizar el puerto PS/2 de la tarjeta Spartan 3E para la obtención de datos en LabView 8.6.
- Utilizar el LCD y los LEDS como indicadores de la encriptación y des encriptación de datos.
- Realizar pruebas con el algoritmo compresión y encriptación de archivos electrónicos del circuito digital implementado en una computadora personal.

1.4.- DATOS DE LA EMPRESA.

La empresa donde se desarrolló el proyecto se describe continuación:

- **Nombre:** Instituto Tecnológico de Tuxtla Gutiérrez
- **Ubicación:** Carretera panamericana Km. 1080 en la ciudad de Tuxtla Gutiérrez, Chiapas, México
- **Código postal:** 29000, apartado postal 599
- **Teléfonos:** (961) 61-5-03-80 y (961) 61-5-04-61
- **Fax:** (961) 61-5-16-87

Misión

Formar de manera integral profesionistas de excelencia en el campo de la ciencia y la tecnología con actitud emprendedora, respeto al medio ambiente y apego a los valores éticos

Visión

Ser una Institución de excelencia en la educación superior tecnológica del Sureste, comprometida con el desarrollo socioeconómico sustentable de la región

Valores

- El ser humano
- El espíritu de servicio
- El liderazgo
- El trabajo en equipo
- La calidad
- El alto desempeño

Las carreras con que cuenta el instituto son:

- Ing. Química
- Ing. Electrónica
- Ing. Eléctrica
- Ing. Mecánica
- Ing. Industrial
- Ing. Bioquímica
- Lic. en Informática
- Ing. en Sistemas Computacionales
- Ing. en Gestión Empresarial.

1.4.1.- Área De Trabajo.

El proyecto se desarrolló en el Laboratorio de Ingeniería Electrónica del Instituto Tecnológico de Tuxtla Gutiérrez, el cual se encuentra ubicado e el edificio I. Esta área se encarga de brindar el material adecuado para realizar pruebas con los diseños de la comunidad tecnológica en los diversos laboratorios con que cuenta.

Para el desarrollo del proyecto se asignó el laboratorio de circuitos impresos, ubicado en el edificio I, cubículo I-15.

1.5.- PROBLEMAS A RESOLVER (PRIORIZÁNDOLOS).

El sistema se realizó en varias etapas, primero se adaptaron los algoritmos que en el FPGA se implementarían, mediante un kit, facilitado por nuestro asesor dentro del ITTG el Dr. Héctor H. de León, una tarjeta SPARTAN 3E de Xilinx, que cuenta con comunicación interfaz USB con la PC.

El principal problema que el sistema tiene se encuentra en la compresión de archivos de texto plano, debido al tamaño del alfabeto que utiliza para comprimir. Con el número de caracteres del alfabeto que este maneja el sistema responde correctamente.

1.6.- ALCANCES Y LIMITACIONES.

En el actual trabajo se pretende desarrollar una arquitectura reconfigurable para la implementación del algoritmo de encriptación AES-128 mas no así el de des-encriptado en la tarjeta Spartan 3E, el programa tiene una interfaz vía USB por LabView 8.6 para que este sea el medio de entrada de los datos para ser transferidos al FPGA en bloques de 128 bits para su encriptación y así mismo sea guardado en un archivo por LabView.

En el transcurso del desarrollo del proyecto surgió la idea de implementar, algoritmos de encriptación y no solamente de compresión de archivos electrónicos, creando de esta manera una alternativa para complementar la codificación de archivos. Este nuevo enfoque del proyecto nos brinda seguridad, para la transmisión de archivos en medios inseguros, por mencionar una de las ventajas más importantes que nos trae la implementación de algoritmos de encriptación.

Con el desarrollo del sistema se pretende implementar en hardware mediante un FPGA y la PC, algoritmos de compresión y encriptación.

La comunicación entre la PC y el FPGA, se implementa mediante el puerto USB por LabView que ya viene implementado en un complemento de FPGA para la tarjeta Spartan 3E. La compresión y encriptación se limita únicamente a archivos de texto plano.

CAPITULO II

2.- FUNDAMENTO TEÓRICO.

2.1.- COMPRESIÓN.

La compresión consiste en reducir el tamaño físico de bloques de información. Un compresor se vale de un algoritmo que se utiliza para optimizar los datos al tener en cuenta consideraciones apropiadas para el tipo de datos que se van a comprimir. Por lo tanto, es necesario un descompresor para reconstruir los datos originales por medio de un algoritmo opuesto al que se utiliza para la compresión. El método de compresión depende intrínsecamente del tipo de datos que se van a comprimir: no se comprime una imagen del mismo modo que un archivo de audio.

2.1.1.- Caracterización De La Compresión.

La compresión se puede definir por el factor de compresión, es decir, el número de bits de la imagen comprimida dividido por el número de bits de la imagen original. El radio de compresión, que se utiliza con frecuencia, es lo contrario al factor de compresión; por lo general, se expresa como porcentaje. Por último, la ganancia de compresión, que también se expresa como porcentaje, equivale a 1 menos el radio de compresión.

2.1.2.- Compresión Simétrica Y Asimétrica

En el caso de la compresión simétrica, se utiliza el mismo método para comprimir y para descomprimir los datos. Por lo tanto, cada operación requiere la misma cantidad de trabajo. En general, se utiliza este tipo de compresión en la transmisión de datos.

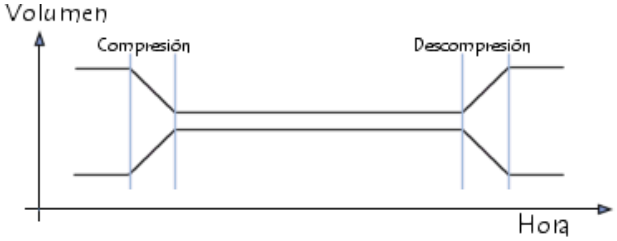


Figura No.1 *"Compresión simétrica"*

La compresión asimétrica requiere más trabajo para una de las dos operaciones. Es frecuente buscar algoritmos para los cuales la compresión es más lenta que la descompresión. Los algoritmos que realizan la compresión de datos con más rapidez que la descompresión pueden ser necesarios cuando se trabaja con archivos de datos a los cuales se accede con muy poca frecuencia (por razones de seguridad, por ejemplo), ya que esto crea archivos compactos.

Existen algunas razones por las que queremos hacer archivos más pequeños:

- Usar menos almacenamiento en disco, en consecuencia menos costos
- Menor tiempo de transferencia
- Mayor velocidad para el acceso secuencial

2.1.3.- Métodos De Compresión.

Los métodos de compresión se pueden clasificar según los algoritmos utilizados, estos pueden ser sin pérdidas ó lossless y con pérdidas ó lossy.

2.1.3.1.- Métodos Sin Pérdidas.

Los métodos sin pérdidas son aquellos métodos, como su nombre indica, que la información descomprimida es exactamente igual a la que había antes de comprimir.

Esto es muy importante tenerlo en cuenta cuando se va a comprimir datos que después se tienen que recuperar y estos no pueden tener ningún dato diferente, ficheros con información en binario, código ejecutable, ficheros de texto, etc.

Algunos ejemplos de algoritmos de compresión sin pérdidas son: Run-Length, Huffman, delta, Lempel-Ziv-Welch los cuales veremos con más detalle a continuación.

2.1.3.2.- Métodos Con Pérdidas.

Los métodos con pérdidas son aquellos que sufren una pequeña modificación al realizar la descompresión, siendo esta la mayoría de las veces imperceptible para la persona normales, las cuales no han desarrollado los sentidos de la vista y el oído como grandes músicos y grandes pintores.

Este método de compresión es utilizado para comprimir imágenes, vídeo y música, ya que, aunque perdamos algún dato no perdemos gran información. Algunos ejemplos de algoritmos de compresión sin pérdidas son: JPEG, MPEG y el MP3.

2.1.4.- Algoritmos De Compresión.

2.1.4.1.- Codificación Run-Length.

Los ficheros de datos frecuentemente contienen una redundancia de caracteres. Por ejemplo, en los ficheros de texto hay muchos espacios en blanco. Debido a esto surgen las primeras ideas para la compresión de datos, ya que si conseguimos agrupar estos conseguiríamos un menor número de caracteres en los ficheros sin modificar el contenido.

A partir de aquí, surge el método de codificación RLE (Run-Length), el cual consiste en mirar la redundancia de un dato, es decir, observar si el dato siguiente es igual que el actual y así sucesivamente hasta que el siguiente cambie, una vez aquí se codifica ese conjunto de datos iguales con el dato y el número de veces seguidas que aparece.

A continuación, vemos un ejemplo que aclarara las cosas:

Dato original **17 8 54 0 0 0 0 0 5 5 5 5 3 4 56 0 0 0 0 ...**

Dato RLE **17 8 54 {0 5} {5 4} 3 4 56 {0 4}...**

Como podemos observar en el ejemplo anterior, hemos conseguido disminuir el tamaño, simplemente haciendo pares de los datos repetidos.

El mayor problema surge a la hora de hacer estas parejas, como saber si es un dato ó el número de datos iguales, para ello se utilizan diferentes métodos, uno de ellos sería el utilizado por en el algoritmo utilizado en PackBits, en el que cada dato, ocho bits es reemplazado por nueve bits, el noveno indica el signo, y así conseguimos tener un rango de valores entre [-255..255], los negativos los utilizaremos para indicar el número de repeticiones y los positivos para indicar el dato.

2.1.4.2.- Algoritmo De Huffman.

El algoritmo de Huffman es un algoritmo que permite comprimir y encriptar datos basándose en la probabilidad de ocurrencia de cada uno de los caracteres contenidos en el mensaje (o archivo) a comprimir/encriptar.

El algoritmo de Huffman asigna a cada carácter del archivo un código (que llamaremos código Huffman) de longitud variable (medida en cantidad de bits). A los caracteres con mayor cantidad de ocurrencias se les asigna un código Huffman con menor longitud y a los que tienen menor cantidad de ocurrencias se les asigna un código con longitud mayor.

Se trata de un algoritmo que puede ser usado para compresión o encriptación de datos.

Este algoritmo se basa en asignar códigos de distinta longitud de bits a cada uno de los caracteres de un fichero. Si se asignan códigos más cortos a los caracteres que aparecen más a menudo se consigue una compresión del fichero.

Esta compresión es mayor cuando la variedad de caracteres diferentes que aparecen es menor. Por ejemplo: si el texto se compone únicamente de números o mayúsculas, se conseguirá una compresión mayor.

Para recuperar el fichero original es necesario conocer el código asignado a cada carácter, así como su longitud en bits, si ésta información se omite, y el receptor del fichero la conoce, podrá recuperar la información original. De este modo es posible utilizar el algoritmo para encriptar ficheros.

Mecanismo del algoritmo:

- Contar cuantas veces aparece cada carácter en el fichero a comprimir. Y crear una lista enlazada con la información de caracteres y frecuencias.
- Ordenar la lista de menor a mayor en función de la frecuencia.
- Convertir cada elemento de la lista en un árbol.
- Fusionar todos estos árboles en uno único, para hacerlo se sigue el siguiente proceso, mientras la lista de árboles contenga más de un elemento:
    - Con los dos primeros árboles formar un nuevo árbol, cada uno de los árboles originales en una rama.
    - Sumar las frecuencias de cada rama en el nuevo elemento árbol.
    - Insertar el nuevo árbol en el lugar adecuado de la lista según la suma de frecuencias obtenida.
- Para asignar el nuevo código binario de cada carácter sólo hay que seguir el camino adecuado a través del árbol. Si se toma una rama cero, se añade un cero al código, si se toma una rama uno, se añade un uno.

Se recodifica el fichero según los nuevos códigos.

Veamos un ejemplo:

Tomemos un texto corto, por ejemplo:

"**ata la jaca a la estaca**"

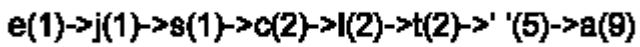
1) Contamos las veces que aparece cada carácter y hacemos una lista enlazada:

' '(5), a(9), c(2), e(1), j(1), l(2), s(1), t(2)

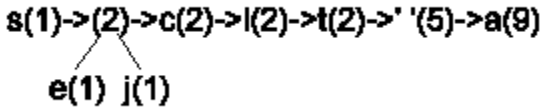
2) Ordenamos por frecuencia de menor a mayor

e(1), j(1), s(1), c(2), l(2), t(2), ' '(5), a(9)

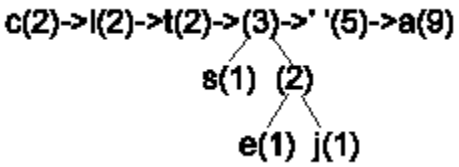
3) Consideremos ahora que cada elemento es el nodo raíz de un árbol.



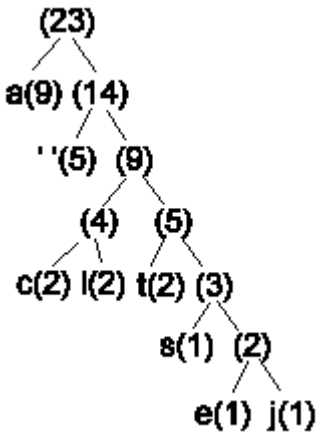
4) Fundimos los dos primeros nodos (árboles) en un nuevo árbol, sumamos sus frecuencias y lo colocamos en el lugar correspondiente:



Y sucesivamente:



El resultado final es:



5) Asignamos los códigos, las ramas a la izquierda son ceros, y a la derecha unos (por ejemplo), es una regla arbitraria.

a	' '	c	l	t	S	e	j
0	10	1100	1101	1110	11110	111110	111111

Tabla No. 1 *"Asignación de código"*

6) Y traducimos el texto:

A	t	a	' '	l	a	' '	j	a	c	a	' '	a	' '	l	a	' '	e	s	t	a	c	a
0	1110	0	10	1101	0	10	111111	0	1100	0	10	0	10	1101	0	10	111110	11110	1110	0	1100	0

Tabla No. 2 *"Traducción del texto"*

Y sólo queda empaquetar los bits en grupos de ocho, es decir en bytes:

01110010	11010101	11111011	00010010	11010101	11110111	10111001	10000000
0x72	0xD5	0xFB	0x12	0xD5	0xF7	0xb9	0x80

Tabla No.3 *"Empaquetamiento en bytes"*

En total ocho bytes, y el texto original tenía 23.

2.1.4.3.- Codificación Delta.

En ciencia, ingeniería y matemáticas, la letra griega delta es usada para denotar el cambio en la variable. El término codificación delta, se refiere pues a la diferencia que existe entre muestras o caracteres consecutivas.

Este método consiste en mantener el primer dato y después tenemos que almacenar en la codificación la diferencia entre el dato actual y el siguiente, el primer dato se mantiene para que en la decodificación podamos recuperar los datos correctamente.

A continuación, mostramos un ejemplo para mejor comprensión del algoritmo:

Dato original **17 19 24 24 24 21 22 25 30 28 ...**

↓

Dato codificado **17 2 5 0 0 -3 1 3 5 -2 ...**

La codificación delta es utilizada cuando entre las muestras no hay mucha diferencia, ya que sino no lograríamos nada de lo que estamos intentando hacer, conseguir que los datos codificados sean lo más iguales posibles, para la posterior utilización de los algoritmos de codificación RLE ó Huffman, que es el método más utilizado por las aplicaciones de compresión de datos.

Los datos que podemos codificar son valores comprendidos en el rango de valores [-127…127], tenemos que codificar tanto números positivos como negativos al calcular la diferencia entre los valores y solo tenemos 8 bits para representarlos y uno tiene que ser para el signo.

2.1.4.4.- Compresión Lempel-Ziv-Welch (LZW).

La compresión LZW debe su nombre a sus desarrolladores, consigue un radio de compresión de hasta el 50%. Este método de compresión es utilizado en imágenes GIF, también en TIFF y PostScript.

LZW utiliza una tabla de códigos para la codificación de los caracteres a sí como cadenas de caracteres, en la tabla hay hasta 4096 entradas de las cuales las 255 primeras son iguales para todos los ficheros comprimidos (valores ASCII), y de la 256 a la 4096 son diferentes según los ficheros que comprimamos.

La tabla de codificación se va creando a medida que se va codificando cada dato con 12 bits, por eso las 4096 entradas, se empieza a codificar a partir de la posición 256 con las combinaciones de caracteres que van apareciendo. Una vez creada la tabla se utiliza para comprimir el fichero.

A continuación, con un ejemplo explicamos el algoritmo:

Código	Caracteres
0	0
1	1
.	.
.	.
254	254
255	255
256	145 201 4
257	243 245
.	.
	.
4095	xxx xxx xxx

Tabla No. 4 *"Códigos"*

Datos originales **123 145 201 4 119 243 245 145 201 4...**

Código de los datos **123 256 119 257 256...**

En principio este algoritmo tiene dos problemas:

- Determinar los códigos, este se soluciona creando la tabla a medida que se leen los datos.
- Como recuperar los códigos, cada fichero tendrá los mismos códigos en las primeras 255 entradas de la tabla, pero el resto no es tan sencillo de determinar. Esto se soluciona añadiendo al fichero esta tabla de códigos a partir de la entrada 256 a la 4095 ya que las anteriores no hacen falta ya que son para todos los ficheros igual.

2.2.- ENCRIPTACIÓN.

2.2.1.- ALGORITMOS DE CLAVE SIMÉTRICA.

Los algoritmos de clave simétrica, también llamados de clave secreta o privada, son los algoritmos clásicos de encriptación en los cuales un mensaje es encriptado utilizando para ello una cierta clave sin la cual no puede recuperarse el mensaje original.

El esquema básico de los algoritmos de clave simétrica es:

Mensaje + clave = código (encriptación)

Código + clave = mensaje (des-encriptación)

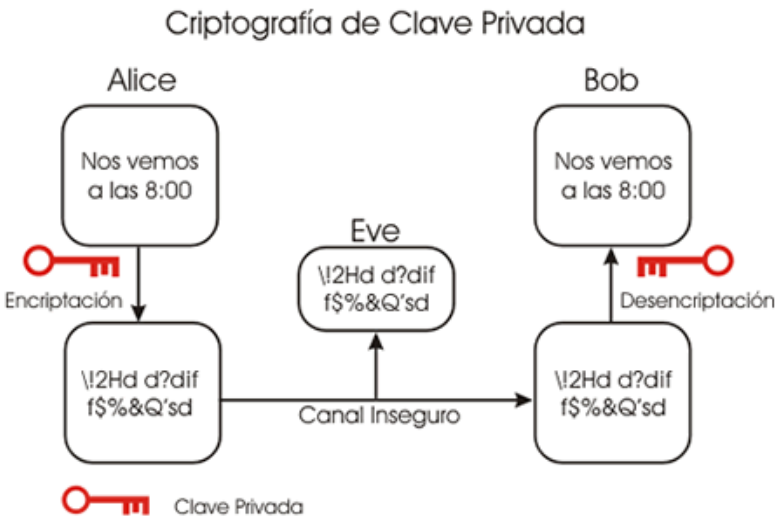


Figura No.2 *"Criptografía de clave privada"*

Esto se lleva a cabo sustituyendo porciones del mensaje original por porciones de mensaje encriptado usando la clave. La sustitución puede ser de varias formas:

Monoalfabética:

Cuando se encripta, cada carácter encriptado corresponde a un carácter del mensaje original y viceversa.

Homofónica:

Cuando un carácter de texto original se encripta en varios caracteres del texto encriptado.

Poligráfica:

Cuando n caracteres del mensaje original generan n caracteres del mensaje encriptado.

Polialfabética:

Cuando n caracteres del texto original se encriptan en m caracteres del texto encriptado (m≠n). Cabe destacar que la sustitución poligráfica y la sustitución homofónica son casos particulares de la sustitución polialfabética.

Sistemas monoalfabéticos y polialfabéticos:

Un algoritmo de encriptación por clave privada es monoalfabético si cada ocurrencia de un mismo carácter en el mensaje original es reemplazada siempre por un mismo carácter en el código cifrado.

Un algoritmo de encriptación por clave privada es polialfabético si cada ocurrencia de un mismo carácter en el mensaje original es reemplazada por distintos caracteres en el código cifrado.

2.2.2.- CRIPTOSISTEMA CAESAR.

El sistema Caesar o desplazamientos Caesar es una de las técnicas de criptografía más simples y mayormente difundidas. Fue el primero que se utilizó del cual se tienen registros. El sistema es monoalfabético y es realmente muy malo, su único valor es el valor histórico de haber sido el primero.

En un sistema Caesar la encriptación se hace por sustitución, cada carácter del mensaje original será reemplazado por un carácter en el mensaje cifrado, el carácter cifrado se obtiene avanzando 'k' pasos en el alfabeto a partir del carácter original. Obviamente 'k' es la clave.

Ejemplo con **k=2**:

Si el texto original es "ABCDE" se codifica como "CDEFG"

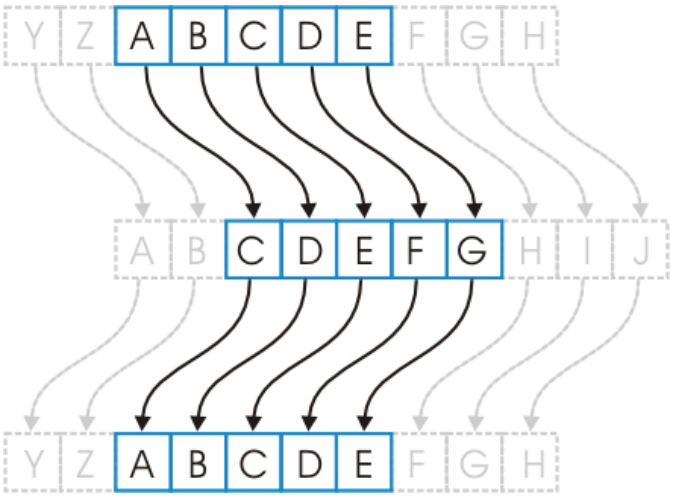


Figura No.3 *"Desplazamiento del abecedario"*

Este es todo el secreto del sistema 'CAESAR' veamos ahora cuan malo es:

Criptoanálisis

Para el sistema Caesar la tarea de un criptoanalista es realmente sencilla, pues la cantidad de posibles claves de este sistema es muy limitada. Trabajando con un alfabeto de 25 caracteres hay solamente 25 posibles claves (1…25) la clave 26, es idéntica a la clave 1, la clave 27 es idéntica a la 2 y así sucesivamente. De esta forma el criptoanalista puede chequear una por una las 25 posibles claves y observando el resultado obtenido se llega fácilmente y en muy poco tiempo al mensaje original.

Este es un criptosistema cuyo punto débil es el espacio de claves, como hay muy pocas claves posibles la técnica mas recomendable para el criptoanalista es simplemente probar todas las posibles claves. A este método se lo denomina 'ataque por fuerza bruta' y cuando el tiempo estimado para el ataque es razonable es un método infalible.

2.2.3.- CRIPTOSISTEMA HILL.

Este sistema está basado en el álgebra lineal y ha sido importante en la historia de la criptografía. Fue Inventado por Lester S. Hill en 1929, y fue el primer sistema criptográfico polialfabético que era práctico para trabajar con más de tres símbolos simultáneamente.

Este sistema es polialfabético pues puede darse que un mismo carácter en un mensaje a enviar se encripte en dos caracteres distintos en el mensaje encriptado. Suponiendo que trabajamos con un alfabeto de 26 caracteres.

Las letras se numeran en orden alfabético de forma tal que A=0, B=1,...,Z=25

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

Tabla No.5 *"Letras numeradas en orden alfabético"*

Se elije un entero d que determina bloques de d elementos que son tratados como un vector de d dimensiones.

Se elije de forma aleatoria una matriz de d × d elementos los cuales serán la clave a utilizar. Los elementos de la matriz de d × d serán enteros entre 0 y 25, además la matriz M debe ser inversible en $\mathbb{Z}_{26}^{n}$.

Para la encriptación, el texto es dividido en bloques de d elementos los cuales se multiplican por la matriz d × d.

Todas las operaciones aritméticas se realizan en la forma modulo 26, es decir que 26=0, 27=1, 28=2 etc. Dado un mensaje a encriptar debemos tomar bloques del mensaje de "d" caracteres y aplicar:

$M \times P_i = C$, donde C es el código cifrado para el mensaje P_i

Criptoanálisis

El sistema de Hill plantea a los criptoanalistas problemas mucho mayores a los que planteaba 'CAESAR'. Para empezar el espacio de claves es mucho mayor, en este caso es de 4C25, es decir las permutaciones de 4 elementos tomados de entre 25 posibles. Y usando una matriz más grande la cantidad de posibles claves se puede hacer tan grande como sea necesario para hacer que sea imposible un ataque por fuerza bruta.

Lo mejor que puede hacer un criptoanalista es tratar de conseguir un código para el cual se conozca una parte del mensaje. Y ver si con ambos datos es capaz de encontrar cual fue la matriz utilizada para encriptar el mensaje.

2.2.4.- CRIPTOSISTEMA PLAYFAIR.

Este sistema criptográfico fue inventado en 1854 por Charles Wheatstone, pero debe su nombre al Baron Playfair de St Andrews quien promovió el uso de este criptosistema. El algoritmo utiliza una matriz de 5x5, como la tabla.

La tabla se llena con una palabra o frase secreta descartando las letras repetidas. Se rellenan los espacios de la tabla con las letras del alfabeto en orden. Usualmente se omite la "W" y se utiliza la "V" en su lugar o se reemplazan las "J" por "I". Esto se hace debido a que la tabla tiene 25 espacios y el alfabeto tiene 26 símbolos. La frase secreta usualmente se ingresa a la tabla de izquierda a derecha y arriba hacia abajo o en forma de espiral, pero puede utilizarse algún otro patrón. La frase secreta junto con las convenciones para llenar la tabla de 5x5 constituye la clave de encriptación.

Por ejemplo:

Si la frase secreta es "CRIPTOSISTEMA PLAYFAIR" llenaremos de izquierda a derecha y arriba hacia abajo y omitiremos la W

C	**R**	**I**	**P**	**T**
O	**S**	**E**	**M**	**A**
L	**Y**	**F**	**B**	**D**
G	**H**	**J**	**K**	**N**
Q	**U**	**V**	**X**	**Z**

Tabla No.6 *"Clave de encriptación"*

La encriptación se realiza de la siguiente forma:

El mensaje original que se desea encriptar es dividido en bloques de dos caracteres cada uno y se le aplican las siguientes cuatro reglas en orden

1. Si en el bloque las dos letras son la misma, se reemplaza la segunda generalmente por una X (o alguna letra poco frecuente) y se encripta el nuevo par.
2. Si las dos letras del bloque aparecen en la misma fila de la tabla, cada una se reemplaza por la letra adyacente que se encuentra a su derecha (si es la letra que se encuentra en la última posición a la derecha de la fila se la reemplaza con la primera de la izquierda de esa fila). Ej. SM se reemplazará por EA y AE por OM.
3. Si las dos letras del bloque aparecen en la misma columna de la tabla, cada una se reemplaza por la letra adyacente que se encuentra por debajo (si es la letra que se encuentra en la última posición inferior de la columna se la reemplaza con la primera de arriba de esa columna). Ej. LC se reemplazará por GO y GQ por QC.
4. Si las letras no se encuentran en la misma fila ni columna se las reemplaza se determina el rectángulo formado por los dos caracteres y se encripta tomando los caracteres que están en las esquinas del rectángulo y en la misma fila que el carácter a encriptar. Ej. SB se reemplazará por MY y KR por HP.

C	R	I	P	T
O	S	E	M	A
L	Y	F	B	D
G	H	J	K	N
Q	U	V	X	Z

Tabla No.7 *"Trasposición de elementos"*

Para des encriptar se aplican estas cuatro reglas en forma inversa, descartando las "X" que no tengan sentido en el mensaje final.

Criptoanálisis:

El sistema Playfair es un sistema de encriptación bastante bueno, la cantidad de posibles claves es enorme ya que son las permutaciones de 25 elementos tomados de entre 26 lo cual da un número muy grande como para derrotar al algoritmo por fuerza bruta. Además es un sistema polialfabético por lo que un análisis de la frecuencia de aparición de cada carácter en el código cifrado no nos aporta nada.

La técnica que se debe utilizar con el esquema Playfair consiste en analizar la frecuencia de aparición de los pares de letras (diagramas) y compararlas con los diagramas más frecuentes del idioma en el cual se supone que se escribió el mensaje original, en castellano los diagramas mas probables son:

Ordenados por frecuencia:

ES, EN, EL, DE, LA, OS, AR, UE, RA, RE, ER, AS, ON, ST, AD, AL, OR, TA, CO

El criptoanalista deberá analizar cuál es el diagrama mas ocurrente en el código cifrado y ver que ocurre si se lo reemplaza por 'ES', de esta forma se van probando distintas combinaciones entre los diagramas más frecuentes en el mensaje cifrado y los diagramas más frecuentes del idioma hasta que se consigue descifrar el texto. Esta es una técnica muy habitual del criptoanálisis y suele funcionar muy bien.

2.2.5.- CIFRADO BÍFIDO.

El método Bífido es un cifrado fraccionario. Es decir que cada letra viene representada por una o más letras o símbolos, y donde se trabaja con estos símbolos más que con las letras mismas.

El método comienza con la utilización de un alfabeto desordenado en una matriz 5x5. Observa un ejemplo donde la clave para el alfabeto desordenado es DIPLOMA:

*	1	2	3	4	5
1	D	IJ	P	L	O
2	M	A	B	C	E
3	F	G	H	K	N
4	Q	R	S	T	U
5	V	W	X	Y	Z

Tabla No.8 *"Alfabeto clave"*

Al ser un cuadro sólo de 5x5 nos vemos obligados a cifrar de la misma forma la I y la J. El contexto nos permitirá distinguir cual de las dos letras se pretendía cifrar. Para cifrar el texto en claro se escriben los equivalentes numéricos de cada letra, utilizando sus "coordenadas". Si por ejemplo el texto en claro es:

VEN A LAS TRES

El equivalente numérico es: **51 25 35 22 14 22 43 44 42 25 43**

Que dividido en dos partes queda: **5 1 2 5 3 5 2 2 1 4 2**

2 4 3 4 4 4 2 2 5 4 3

Si ahora leemos los números como columnas en lugar de por filas resulta: **52 14 23 54 34 54 22 22 15 44 23**

Que volviendo a consultar la tabla resulta en el mensaje cifrado:

WLBYKYAAOTB

Este método altera la frecuencia de los caracteres a diferencia de lo que ocurre por ejemplo con los cifrados monoalfabéticos. Admite algunas variaciones como por ejemplo dividir la lista en 3, 4,...n partes

2.1.6.- EL CIFRADO DE GRONSFELD.

El cifrado de Gronsfeld es del tipo conocido como polialfabéticos. Esto significa que se usa más de un alfabeto cifrado para poner en clave el mensaje y que se cambia de uno a otro según se pasa de una letra del texto en claro a otra.

Es decir que deben tenerse un conjunto de alfabetos cifrados y una forma de hacer corresponder cada letra del texto original con uno de ellos. Para dejar esto más claro veamos una de las tablas que se usaban antes de la era de los ordenadores para hacer un cifrado de este tipo.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0:	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
1:	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
2:	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
3:	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
4:	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
5:	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
6:	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
7:	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
8:	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
9:	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B

Tabla No.9*"Alfabetos para el cifrado Gronsfeld"*

Ahora cogemos una serie de dígitos como clave. Supongamos que usamos 1203456987. Vamos a utilizar la tabla para el cifrado del mensaje:

EMBARCAMOS AL ANOCHECER

Empezamos por escribir la clave debajo del texto original las veces que sea necesario:

EMBARCAMOS AL ANOCHECER

1203456987 12 034569871

Y sustituimos cada letra por la correspondiente del alfabeto que indica el número debajo de él.

HRDHCPROLL DQ CUZPYGZXU

El cifrado de Gronsfeld altera la frecuencia de las letras del texto, pues por ejemplo la letra más corriente en castellano, la E, se cifra de forma diferente según su posición en el texto original.

2.1.7.- EL CIFRADO ADFGVX.

Este cifrado apareció por primera vez al final de la primera guerra mundial, antes de lo que iba a ser la ofensiva definitiva del ejército alemán. La fortaleza del método estribaba en que era uno de los primeros en unir transposición y sustitución, dos procesos que producen, en la terminología actual, difusión y confusión.

Según explica Simon Singh en "Los códigos secretos", fue descifrado por primera vez por un francés, Georges Painvin, que trabajo día y noche perdiendo quince kilos en el proceso.

Se empieza disponiendo las 26 letras del alfabeto anglosajón y los diez dígitos en una matriz 6x6. Las líneas y las columnas van encabezados por las letras A D F G V X. El modo de ordenar letras y números en la cuadrícula forma parte de la clave y necesita ser comunicada al receptor del mensaje. Pongamos por ejemplo:

	A	D	F	G	V	X
A	0	q	9	z	7	c
D	m	u	1	h	f	2
F	4	8	w	n	r	g
G	l	6	v	t	p	a
V	y	3	d	5	e	k
X	j	s	i	o	b	x

Tabla No.10 *"Cuadricula clave"*

En primer lugar tomaremos cada letra del mensaje en claro sustituyéndola por las letras correspondientes a su fila y columna. Por ejemplo el número 5 sería sustituido por las letras VG y la j por el par de letras XA.

Pongamos como ejemplo el mensaje: Envíen municiones

e	n	v	i	e	n	m	u	N	i	c	I	o	n	e	s
VV	FG	GF	XF	VV	FG	DA	DD	FG	XF	AX	XF	XG	FG	VV	XD

Tabla No.11 *"Sustitución de caracteres"*

Hasta aquí solo un cifrado ordinario por sustitución que se descifra con un análisis de frecuencia si se dispone de suficiente texto. Sigue otra frase con una transposición dependiente de una palabra clave. Supongamos que la clave es WHISKY. Las letras de la clave se escriben en la cabecera de una cuadrícula.

El texto que hemos cifrado antes se escribe por filas en dicha cuadrícula así:

W	H	I	S	K	Y
V	V	F	G	G	F
X	F	V	V	F	G
D	A	D	D	F	G
X	F	A	X	X	F
X	G	F	G	V	V
X	D	A	A	A	A

Tabla No.12 *"Palabra clave"*

Donde hemos añadido dos caracteres de relleno (00 ~ AA AA) para que el cuadro quede completo. Ahora las columnas de la cuadrícula se cambian de posición de modo que las letras de la clave queden en orden alfabético:

H	I	K	S	W	Y
V	F	G	G	V	F
F	V	F	V	X	G
A	D	F	D	D	G
F	A	X	X	X	F
G	F	V	G	X	V
D	A	A	A	X	A

Tabla No.13 *"Ordenamiento alfabético de columnas"*

Para acabar leemos por columnas la cuadrícula y el resultado es el texto cifrado, en este caso:

VFAFGDFVDAFAGFFXVAGVDXGAVXDXXXFGGFVA

El inconveniente de este tipo de encriptación, es que entrega el doble de caracteres a encriptar, es decir, si tenemos un texto con 10 caracteres, al aplicar este algoritmo nos devuelve un texto encriptado el cual posee 20 caracteres.

2.3. - ALGORITMO AES (Advanced Encryption Standard).

Historia del algoritmo

El 23 de Noviembre de 1976 se establece el primer estándar de cifrado para la comunicaciones, el denominado a **algoritmo DES**. Desde entonces se han descrito multidtud de ataques que permiten criptoanalizarlo más rápidamente que con un ataque por fuerza bruta. Pero sin duda la publicación de la asociación EFF (Electronic Frontier Fundation) del diseño de una máquina que permite atacarlo por fuerza bruta en un tiempo infinito (DES-CRACKER), puso el grito en el cielo de la comunidad científica. El estándar había consumido su tiempo de vida.

En el año 1997, el Instituto Nacional de Estándares y Tecnología de EEUU (NIST), emprende un proceso abierto para la selección de un nuevo algoritmo de cifrado, que sustituya al actual estándar de cifrado, criticado por especialistas e instituciones en seguridad.

Este nuevo algoritmo sería útil no sólo para proteger la información del Gobierno EEUU, sino que también seria utilizado masivamente por el sector privado, y adoptado como estándar por el resto de países, entre ellos los Europeos.

En Septiembre de 1997 se presentan los criterios de evaluación y requisitos mínimos que debían cumplir todos los algoritmos que optarán a ganar el concurso, entre ellos destacaban:

- El algoritmo debe ser público.
- Debe ser un algoritmo de cifrado en bloque simétrico.
- La longitud de la clave debe ser como mínimo 128 bits.
- Su diseño debe permitir aumentar la longitud de la clave según las necesidades.
- Debe ser implementable tanto en H¨W como en SW

Los algoritmos que cumplieran los requisitos anteriores serían juzgados por los siguientes factores:

- Seguridad.
- Eficiencia Computacional.
- Requisitos de Memoria.
- Implementable en HW y SW.
- Simplicidad de Diseño
- Flexibilidad.

Los algoritmos que se presentaron a este concurso además tenían que soportar obligatoriamente una longitud de bloque de 128 bits como mínimo, y una longitud de clave de 128, 192 y 256 bits, al margen de cualesquiera otras longitudes posibles.

Para llevar a cabo la elección del algoritmo se propuso crear dos rondas de selección. En la primera ronda se seleccionaría los 5 algoritmos mejores, que cumplieron las especificaciones iniciales, y en la segunda ronda se decidiría el algoritmo ó algoritmos ganadores.

El 2 de octubre de 2000, el NIST anunció el algoritmo ganador. Las votaciones del concurso establecieron el siguiente ranking:

RINJDAEL	86 VOTOS
SERPENTE	59 VOTOS
TWOFISH	31 VOTOS
RC6	23 VOTOS
MARS	13 VOTOS

El algoritmo Rijndael ganó el concurso, por permitir la mejor combinación de seguridad-velocidad-eficiencia, sencillez y flexibilidad. Y será el algoritmo de encriptación que en este trabajo usaremos para lograr nuestro objetivo, mostrando en líneas mas adelante los fundamentos matemáticos que hacen posible su comprensión y uso.

Comparación de los Algoritmos

LA seguridad de los algoritmos, fue el aspecto más importante se tuvo en cuenta en la segunda ronda del proceso de selección del NIST, para probar la seguridad de cada algoritmo se lanzaron varios ataques a cada algoritmo, estos ataques han sido llevados a cabo, tanto por los autores como por los autores de algoritmos rivales, como por cualquier otra persona que enviara la documentación pertinente al NIST.

En cualquiera de los casos, según los ataques publicados, los resultados obtenidos en cada algoritmo fueron:

- MARS: parece tener un margen de seguridad elevado. MARS recibió críticas basadas en su complejidad, la cual puede haber obstaculizado sui análisis de seguridad durante el proceso de desarrollo del AES.
- RC6: parece tener un margen de seguridad adecuado. Sin embargo, RC6 recibió alguna crítica debido a su bajo margen de seguridad respecto al que ofrecieron otros finalistas. Por otro lado, RC6 ha sido elogiado por su simplicidad, la cual ha facilitado su análisis de seguridad durante el tiempo especificado en el proceso de desarrollo del AES.
- RIJNDAEL: parece tener un margen de seguridad adecuado. El margen de seguridad es un poco difícil de medir, debido a que el número de rondas cambia con el tamaño de la clave. Rijndael recibió críticas sobre su margen de seguridad, ya que es de los más bajos, entre los finalistas, y que su estructura matemática puede conducir a ataques. Sin embargo, su estructura es bastante simple, esto ha facilitado su análisis de seguridad durante el tiempo especificado en el proceso de desarrollo del AES.
- SERPENT: parece tener un margen de seguridad alto. Serpent también tiene una estructura simple, que ha facilitado su análisis de seguridad durante el tiempo especificado de desarrollo del AES.

- TWOFISH: parece tener un margen de seguridad alto. El concepto de margen de seguridad tiene menos significado para este algoritmo que par los demás finalistas. La dependencia de las S-cajas de Twofish en solo K/2 bits de entropía en el caso de clave K-bits ha producido algunas especulaciones acerca de que Twofish puede ser sensible a un ataque divide y vencerás, aunque tal ataque no ha sido comprobado. Twofish ha recibido alguna crítica por su complejidad, haciendo difícil su análisis durante el tiempo establecido en el proceso de desarrollo del AES.

Otro tema muy importante fue el estudio de la velocidad con la que se ejecutaba cada algoritmo, por ejemplo, dependiendo del tamaño de la clave. La realización de SW de MARS, RC& y SERPENT no varían significativamente para los tres tamaños de clave de AES. Para Rijndael y Twofish, sin embargo, la configuración de la clave es lógicamente, más lento para claves de 192 bits que para claves de 128 bit, y más lento todavía para claves de 256 bits, aunque en estos casos ofrecen una compresión en el incremento de la seguridad.

Rijndael se presentó como el algoritmo más rápido en multitud de plataformas: 32 bit, procesadores de 8bits, etc.

2.3.1.- PRINCIPIOS ALGEBRAICOS BÁSICOS.

En este capítulo se resumen los aspectos más sobresalientes de la Teoría de Campos Finitos. Se presentan primero conceptos fundamentales del álgebra de Grupos, Anillos y Campos.

A continuación se examina la estructura de Campos Finitos. Por último se analizan los principios básicos de Campos Compuestos por hallarse este tema íntimamente relacionado con la implementación del Algoritmo Rijndael.

2.3.2.- GRUPOS.

Definición A: Un conjunto de *G* elementos sobre los cuales se ha definido una operación binaria, se denomina *grupo* cuando satisface condiciones de asociatividad, contiene un elemento identidad único y, para cada elemento de *G*, existe un único elemento inverso. Si la operación binaria satisface condiciones de conmutatividad se dice que *G* es un *grupo conmutativo o abeliano.* Entre los grupos conmutativos más conocidos se encuentran el de los números enteros y el de los números racionales. El conjunto de enteros es un grupo conmutativo bajo la operación de suma (+), siendo el *0* el elemento identidad y *–i* la inversa de un número entero *i*. El conjunto de números racionales excluyendo el cero es un grupo conmutativo bajo la operación de multiplicación (*x*), siendo *1* el elemento neutro y el número *b/a* el inverso multiplicativo de *a/b.* El conjunto *G = {0, 1}* bajo la operación suma *módulo-2* (⊕) es un grupo conmutativo. La operación es equivalente en el Algebra Booleana a una *EXOR*. En dicho grupo, *0* es el elemento identidad y la inversa de cada elemento es el mismo elemento.

Definición B: El número de elementos de un grupo *G* se denomina *orden del grupo* y se denota *ordG*. De esta manera, según la cantidad de elementos, se puede hablar de dos grandes clases de grupos: *infinitos* y *finitos*. Para los fines de esta tesis, se consideran sólo los grupos finitos.

Definición C: El orden de un elemento *g* en un grupo finito *G*, es el número más pequeño $s > 0$, tal que g^s tenga como resultado el elemento identidad y se denota *ordg*. Se deduce que existirán *s* potencias distintas de *g*. Se deduce también que el orden máximo de un elemento no puede superar al orden del grupo al que pertenece, pues entonces el conjunto de potencias de dicho elemento sería mayor que el número de elementos del grupo. Dado que todos los elementos de *G* deben ser distintos entre sí, esta última situación no sería posible.

Definición D: Siendo *m* un entero positivo, es posible definir un grupo finito conmutativo de orden *m*: *G = { 0, 1, 2, ... ,m-1}*, bajo la operación *suma módulo-m*, siendo *0* el elemento identidad y *(m-i)* el elemento inverso de *i*. Se lo conoce como *grupo aditivo*.

Definición E: El grupo de enteros *G = {1, 2, 3,...., p-1}*, con *p* primo, bajo la operación *multiplicación módulo-p* es un grupo conmutativo cuyo elemento identidad es *1*. Siendo *i* un elemento de *G* tal que $i < p$, *i* y *p* deben ser primos relativos por lo que se verifica que existen dos enteros *a* y *b* que cumplen el *Algoritmo de Euclides*. Por el *Teorema de Euclides a* y *p* son primos relativos y se relacionan por medio de la Ec.01.

$$a.i + b.p = 1 \qquad \text{(Ec.01)}$$

Se verifica que, si el producto (*a.i*) se divide por *p*, el resto de la división da *1*. Si 0 < *a* < *p*, *a* pertenece a *G* y resulta ser inverso multiplicativo de *i*.

Sin embargo, si *a* no pertenece a *G*, al dividir *a* por *p*, el resto *r* < *p* es distinto de *p* y de *0* pues *a* y *p* son primos relativos. De este modo *r* pertenece a *G* y es tal que 1<*r* <(*p* -1) y resulta ser el inverso multiplicativo de *i*. Las Ec.02 a Ec.05 ilustran lo explicado.

$$a = qp + r \quad \text{(Ec.02)}$$

$$a.i = qp.i + r.i \quad \text{(Ec.03)}$$

$$qp.i + r.i = -b.p + 1 \quad \text{(Ec.04)}$$

$$r.i = (-b + q.i)\, p + 1 \quad \text{(Ec.05)}$$

El grupo *G = {1, 2, 3,...., p-1}* bajo la *multiplicación módulo-p* resulta ser un *grupo multiplicativo*. Si *p* no es un número primo, *G* no es grupo bajo la *multiplicación módulo-p*.

Definición F: Se dice que *H* es subgrupo de *G*, si *H* es subconjunto de *G* cerrado bajo la operación de *G* y satisface todas las condiciones de un grupo. Si *a* es un elemento de un grupo, el conjunto de todas las potencias enteras de *a* es un subgrupo. Por ejemplo el conjunto de todos los números enteros es un subgrupo del grupo de números racionales bajo la operación de suma de números reales.

Definición G: Se dice que un grupo es *cíclico* si para algún elemento *a* que pertenece al grupo, todo elemento que pertenezca al grupo es de la forma *am*, con *m* perteneciente a los enteros. El elemento *a* se llama *generador* del grupo y es tal que *orda = ordG*. Todo grupo cíclico es *abeliano*. Un elemento *at* de un grupo cíclico finito de orden *n* es generador del grupo si se verifica *(t, n) = 1*, o sea *t* y *n* son primos relativos.

Definición H: Se denomina *Sn* al conjunto de las *n!* permutaciones de *n* símbolos. Este conjunto resulta ser un grupo respecto de la operación permutación (°). Como dicha operación no es conmutativa, un grupo de permutaciones es no abeliano.

Definición I: El teorema de *Lagrange* establece que el orden de cada subgrupo de un grupo finito *G* es divisor del orden de *G*. Por tanto si *G* es finito, el orden de cualquier elemento *a* □□*G* (o sea el orden del subgrupo cíclico generado por *a*) es divisor de *n*. También todo grupo de orden primo es cíclico.

2.3.3.- CAMPOS.

Definición A: Sea un conjunto de elementos *F* sobre el cual se definen dos operaciones binarias denominadas adición (+) y multiplicación (.), *F* es un *campo* si se verifica:

- *F* es *grupo conmutativo* bajo la (+) con elemento identidad *0*.
- El conjunto de elementos distintos de *0* de *F* es un *grupo multiplicativo* bajo la (.) con elemento identidad *1*.
- La operación multiplicación (.) es distributiva respecto de la suma (+).

El número de elementos de un campo se denomina *orden*. Los campos finitos se conocen como *campos de Galois* y se denotan como *GF* (*Galois Field*) en honor a su descubridor.

Definición B: Para cada elemento *a* perteneciente al campo *F* existe un inverso aditivo - *a* y, si *a* es distinto de *0*, también existe un inverso multiplicativo *a*-1.

El conjunto *{0, 1}* es *grupo conmutativo* bajo la *adición módulo-2* y *{1}* es *grupo multiplicativo* bajo la *multiplicación módulo-2*. Se puede verificar en este caso el cumplimiento de la propiedad distributiva, con lo cual *{0, 1}* es un campo de dos elementos bajo las operaciones mencionadas. Este campo se denomina *campo binario*, se denota *GF(2)* y juega un rol muy importante en la *Teoría de Códigos en Comunicaciones Digitales*.

El conjunto *{0, 1, 2,...., p-1}* es un campo de orden *p* bajo la *adición* y la *multiplicación módulo-p*, y se lo denota *GF(p)*. Para cualquier número primo *p* existe un campo finito de orden *p*. De hecho, para cualquier entero positivo *m*, es posible extender el campo *GF(p)* a un campo de p^m elementos que se denomina *campo extendido de GF(p)* y se denota *GF(p^m)*. Se ha demostrado que el orden de cualquier campo finito es una potencia de un número primo, p^m .

Definición C: Dado un campo finito de *q* elementos *GF(q),* el menor entero positivo λ, tal que $\sum_{i=1}^{\lambda} 1 = 0$ se conoce como *característica* de *GF(q)* y siempre es un número primo. De esta definición se sigue que, dados dos números enteros positivos *k* y *m*, menores que λ, se verifica la relación que se presenta en Ec.06.

$$\sum_{i=1}^{k} 1 \neq \sum_{i=1}^{m} 1 \qquad \text{(Ec.06)}$$

La sucesión de diferentes sumatorias hasta llegar a λ, genera λ elementos diferentes de *GF(q)* y el conjunto de las mismas es en sí mismo un campo

de λ elementos *GF(λ)*, *subcampo* de *GF(q)*. Se puede probar que si $q \neq \lambda$, entonces q es potencia de λ.

Se resaltan de este modo dos elementos importantes relativos a este tema:

- La *característica* de cualquier campo finito es un número primo.
- Todo campo finito de *característica p* contiene el campo de enteros *modulo-p* como *subcampo*.

Definición D: Dado un elemento a distinto de *0*, perteneciente a *GF(q)*, si se toma la secuencia de potencias de a, se verifica que existen dos números positivos k y m, con $m > k$, tales que $a^k = a^m$. Si $a - 1^{-1}$ es el inverso multiplicativo de a, a^{-k} es el inverso multiplicativo de a^k, de tal modo que es factible verificar la Ec.07:

$$1 = a^{m-k} \qquad \text{(Ec.07)}$$

Esta consideración implica que existe un entero positivo n, tal que $a^n = 1$ y todas las potencias de a, hasta la potencia *n-ésima* son diferentes entre sí. El *orden* de un elemento a es el entero n y las potencias mencionadas forman un *grupo cíclico* bajo la operación de multiplicación de *GF(q)*.

Definición E: Si a es cualquier elemento distinto de cero de *GF(q)*, entonces $a^{q-1} = 1$ y si el orden de a es n, n divide a *q-1*. En un campo finito *GF(q)*, se dice que un elemento a distinto de cero, es *primitivo* si el orden de a es *q-1* .

2.3.4.- ARITMÉTICA DE CAMPO BINARIO.

Los códigos y técnicas más utilizados en transmisión y almacenamiento cifrado de datos pertenecen a *GF(2)* o sus extensiones *GF(2m)*. Se debe, entonces, profundizar el conocimiento y la interpretación de los datos como pertenecientes a alguno de los campos mencionados y, por consiguiente, de las operaciones a las que se ven sujetos. De este modo será posible comprender mejor diferentes posibilidades de funcionamiento y de implementación de los diversos diseños de circuitos utilizados para aplicar el procesamiento requerido.

Definición A: Un polinomio *f(x)* sobre un campo *F* se puede expresar según la Ec.08:

$$f(X) = f_0 + f_1 X + f_2 X^2 + ... + f_n X^n \qquad \text{(Ec.08)}$$

donde cada uno de los coeficientes f_i es un elemento de *F*. *X* se denomina indeterminado y *n* es el grado del polinomio. El grado es el mayor número entero tal que $f_n \neq 0$. Los polinomios que verifican $f_n = 1$ se denominan *polinomios mónicos*.

Se considera trabajar con polinomios cuyos coeficientes pertenecen a *GF(2)*, a los que se denomina *polinomios sobre GF(2)*. Se trata de aquellos polinomios para los que se verifica $f_i = 0$ ó 1, en el caso $0 \leq i \leq n$. Por ejemplo, existen dos polinomios de grado 1, *X* y 1+ *X*, cuatro polinomios de grado 2, X^2, $1 + X^2$, $X + X^2$ y $1 + X + X^2$. En general existen 2^n polinomios sobre *GF(2)* de grado *n*. Sobre cualquiera de ellos se puede operar con suma, resta, multiplicación y división. En la multiplicación y adición de coeficientes se considera la *aritmética módulo-2* (*EXOR*) y se puede demostrar que la suma es conmutativa, asociativa y distributiva con respecto al producto. Por otra parte, al dividir dos polinomios entre sí se obtiene un polinomio cociente y otro polinomio resto. Si este último es cero se dice que los dos polinomios son divisibles.

Por otra parte, si *a* es raíz de un polinomio, dicho polinomio es divisible por *X* + *a*. Para polinomios sobre *GF(2)* que tienen un número par de términos, todos son divisibles por *X* +1.

Definición B: Un polinomio de grado *m* se dice que es *irreducible* sobre *GF(2)* si no es divisible por ningún polinomio sobre *GF(2)* de grado menor que *m* pero mayor que *0*. Es decir que el concepto de irreducible para polinomios es análogo al concepto de primalidad para enteros.

Un teorema importante referido a los polinomios irreducibles sobre *GF(2)* establece que un polinomio irreducible sobre *GF(2)* de grado *m* divide a $x^{2^m-1} + 1$.

Dos importantes consecuencias de la definición de polinomios *irreducibles* son:

- Para cada polinomio irreducible *f(x)* sobre el campo *F*, existe un polinomio Mónico irreducible *g(x)*, tal que $f(x) = a.g(x)$, siendo *a* un escalar, de tal modo que se cumple: $g(x) = a^{-1} f(x)$.

- Si *f(x)* es irreducible sobre el campo *F*, no necesariamente es irreducible sobre otro campo *G*.

Definición C: Si *f(x)* es un polinomio irreducible de grado *m* sobre *GF(p)* y β una raíz de dicho polinomio, claramente β no pertenece a *GF(p)*. El conjunto de polinomios grado < *m* en *GF(p)[β]*, junto con la multiplicación y adición módulo *f*(β), forma un campo finito único respecto al isomorfismo.

Se trata de una *extensión* de *GF(p)* con p^m elementos, *GF(p^m)*. Esencialmente esta definición establece que existe un único campo finito con p^m elementos, para cualquier primo p y un entero m.

Definición D: Se dice que un elemento a perteneciente a un campo F es un *elemento primitivo* de F si cualquier elemento distinto de cero de F se puede expresar como una potencia de a. Un polinomio irreducible sobre el subcampo primo de F que tenga un elemento primitivo de F como raíz se denomina *polinomio primitivo*. Un polinomio irreducible de grado m se verifica que es *primitivo* si el menor entero positivo n para el cual dicho polinomio divide a X^n +1 es $n = 2m - 1$.

Existen tablas de polinomios primitivos donde se señalan los irreducibles.

Un polinomio con coeficientes en *GF(2)* puede tener raíces que pertenezcan a un campo extensión de *GF(2)*. Por ejemplo el polinomio $X^4 + X^3 + 1$ es irreducible sobre *GF(2)* por lo que no posee raíces en *GF(2)*, aunque sí las posee en *GF(2^4)*, siendo estas α^7, α^{11}, α^{13} y α^{14}.

Se puede demostrar que, siendo *f(X)* un polinomio con coeficientes en *GF(2)* y β un elemento del campo extendido de *GF(2)*, si β es raíz de *f(X)*, entonces para cualquier $l \geq 0$, β^{2^l} también raíz de *f(X)*. Como β es un elemento de *GF(2^m)* se verifica que $\beta^{2^m-1} = 1$. Esta relación también se puede expresar como $\beta^{2^m-1} = 0$. De este modo, se cumple que β es raíz del polinomio $X^{2^m-1} + 1$ cuyo grado es $2^m - 1$. Es decir que los $2^m - 1$ elementos de *GF(2^m)* son las raíces de dicho polinomio. Por otro lado, todos los elementos de *GF(2^m)*, incluyendo el *0*, son raíces de $X^{2^m} + X$.

Debido a lo anteriormente expresado, β puede ser raíz de un polinomio en *GF(2)* con grado menor a 2^m. Sea φ*(X)* el polinomio de menor grado sobre *GF(2)* para el que se verifica φ*(β) = 0*. Este polinomio único se conoce como *polinomio mínimo* de β. Se puede demostrar que dicho polinomio es irreducible. También se puede demostrar que, si β es raíz de *f(X)*, entonces *f(X)* es divisible por φ*(X)*. El polinomio mínimo también divide a $X^{2^m} + X$.

Siendo β un elemento en *GF(2^m)* y e el menor entero positivo tal que $\beta^{2^e} = \beta$, se verifica la Ec.09:

$$f(X) = \prod_{i=0}^{e-1} (X + \beta^{2^i}) \qquad \text{(Ec.09)}$$

De este modo, *f(X)* es un polinomio irreducible sobre *GF(2)*, que resulta ser el polinomio mínimo φ*(X)*. En particular, el grado de un polinomio mínimo de cualquier elemento en $GF(2^m)$ divide a *m*.

2.3.5.- CONSTRUCCIÓN DE GF(2^m).

Un método para construir un campo $GF(2^m)$ a partir de *GF(2)*, consiste en comenzar con dos elementos *0* y *1* de *GF(2)* y un nuevo símbolo α para armar la secuencias de potencias de α a partir de la definición de la multiplicación (.). La Ec.10 representa los elementos de dicho campo.

$$
\begin{aligned}
&0.0 = 0\\
&0.1 = 1.0 = 0\\
&1.1 = 1\\
&0.\alpha = \alpha.0 = 0\\
&1.\alpha = \alpha.1 = \alpha\\
&\alpha^2 = \alpha.\alpha\\
&\alpha^3 = \alpha.\alpha.\alpha\\
&\quad\vdots\\
&\alpha^j = \alpha.\alpha.....\alpha(jveces)\\
&\quad\vdots
\end{aligned}
\qquad \text{Ec.10)}
$$

Se define un conjunto de elementos *F* que consiste en la serie de potencias de anteriormente presentada y la restricción que *F* contenga sólo 2^m elementos. Si *p(X)* es un polinomio primitivo de grado *m* sobre *GF(2)* y se supone que verifica *p*(α) = 0 , dado que *p(X)* divide a $X^{2^m} + 1$, al reemplazar *X* por α resulta:

$$X^{2^m-1}+1 = p(X).q(X) \qquad \text{(Ec.11)}$$

$$\alpha^{2^m-1}+1 = p(\alpha).q(\alpha) \qquad \text{(Ec.12)}$$

$$\alpha^{2^m-1}+1 = 0.q(\alpha) \qquad \text{(Ec.13)}$$

$$\alpha^{2^m-1} = 1 \qquad \text{(Ec.14)}$$

Es decir que, bajo la condición $p(\alpha) = 0$, el conjunto F se convierte en un campo finito de elementos $F^* = \{ 0, 1, a, \alpha^2 , ..., \alpha^{2^m-2}\}$ que a su vez es un grupo conmutativo de orden $2m-1$ bajo la multiplicación si se excluye el elemento 0.

Se pretende definir una operación adición sobre F^* para que también se verifique la propiedad de grupo conmutativo. En el caso en que $0 \le i \le 2^m - 1$, al dividir el polinomio X^i por $p(X)$ se obtiene:

$$X^i = q_i(X)\,p(X) + a_i(X) \qquad \text{(Ec. 15)}$$

$$a_i(X) = a_{i0} + a_{i1}X + a_{i2}X^2 + ... + a_{im-1}X^{m-1} \qquad \text{(Ec. 16)}$$

X y $p(X)$ son primos relativos. X^i no es divisible por $p(X)$ y se verifica que, para cualquier $i \ge 0$, es $a_i(X) \neq 0$. También se puede demostrar que, para $i \le 0$,$j<2^m - 1$, $i \neq j$, $a_i(X) \neq a_j(X)$. De esta manera se obtienen $2^m - 1$ polinomios $a_i(X)$ distintos de grado $m-1$ o menor. Si se reemplaza X por α se verifica la Ec.17:

$$\alpha^i = a_i(\alpha) = a_{i0} + a_{i1}\alpha + a_{i2}\alpha^2 + ... + a_{im-1}\alpha^{m-1} \qquad \text{(Ec. 17)}$$

Se ve que $2^m - 1$ elementos distintos de cero, $\alpha^0 \quad \alpha^1 , ..., \alpha^{2^m-2}$ en F^*, se representan por $2^m - 1$ *polinomios distintos de cero* de α sobre $GF(2)$ con grado $m-1$ o menor.

Definida la (+), se puede comprobar que F^* es un grupo conmutativo, en el que 0 es la identidad y cada elemento es inverso de sí mismo. También los elementos no nulos de F^* forman un grupo conmutativo respecto de la multiplicación (.). Si se usa la representación polinomial para los elementos de F^* se verifica la distributividad de la multiplicación sobre la suma, con lo cual F^* es un *campo de Galois* de 2^m elementos, $GF(2^m)$, uno de cuyos subcampos es $GF(2)$ y la característica es 2. La representación por potencias es útil para las operaciones que impliquen una multiplicación, mientras que la representación de polinomios lo es para aquellas que se basen en sumas.

Ejemplo. Siendo m = 4 y el polinomio $p(X)$ primitivo sobre $GF(2)$, $p(x)=1+X+X^4$, si se verifica α como raíz, resulta $p(\alpha) =1+\alpha+\alpha^4= 0$ ó $1+ \alpha = \alpha^4$. Con estos supuestos podemos construir $GF(2^4)$:

Representación en potencias	*Representación polinomial*	*Representación por 4-úpla*
0	0	(0 0 0 0)
1	1	(1 0 0 0)
α	α	(0 1 0 0)
α^2	α^2	(0 0 1 0)
α^3	α^3	(0 0 0 1)
α^4	$1+\alpha$	(1 1 0 0)
α^5	$\alpha+\alpha^2$	(0 1 1 0)
α^6	$\alpha^2+\alpha^3$	(0 0 1 1)
α^7	$1+\alpha+\alpha^3$	(1 1 0 1)
α^8	$1+\alpha^2$	(1 0 1 0)
α^9	$\alpha+\alpha^3$	(0 1 0 1)
α^{10}	$1+\alpha+\alpha^2$	(1 1 1 0)
α^{11}	$\alpha+\alpha^2+\alpha^3$	(0 1 1 1)
α^{12}	$1+\alpha+\alpha^2+\alpha^3$	(1 1 1 1)

Tabla No.14 *Elementos de $GF(2^4)$ generados por $p(x)=1+X+X^4$.*

2.3.5.1- Campo GF(2^8).

Operaciones con palabras de 8 bits

Varias operaciones presentes en el algoritmo *Rijndael* son de la forma orientada al *byte*. De este modo, los conjuntos de 8 bits pueden ser interpretados como elementos de $GF(2^8)$: polinomios cuyo grado máximo es *7* y cuyos coeficientes pertenecen a *GF(2)*. Otras operaciones del algoritmo involucran palabras de *4 bytes* que se corresponden con polinomios de grado menor a *4* y coeficientes en $GF(2^8)$.

La diversas representaciones del campo $GF(2^8)$ son isomórficas, pero a pesar de esta quivalencia, la complejidad de la implementación del algoritmo depende fuertemente de la representación elegida.

La Ec.(3.18) es la representación de un byte *b*, consistente de la cadena de bits $b_7\, b_6\ b_5\ b_4\ b_3\, b_2\, b_1\, b_0$, como un polinomio con coeficientes en *{0, 1}*.

$$b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0 \qquad \text{(Ec.18)}$$

La suma de elementos se corresponde con la suma de elementos en *GF(2)*. Su resultado es un polinomio cuyos coeficientes resultan de la *suma módulo-2* de los coeficientes respectivos. Se cumplen así todas las condiciones de un *grupo Abeliano*.

Por ejemplo, el byte cuya representación binaria es (*01010111*) y cuya representación hexadecimal se corresponde con (*0x57*) tiene la representación polinomial dada por $x^6 + x^4 + x^2 + x + 1$. Con esta representación, la suma de los bytes (*0x57*) y (*0x58*) es el polinomio: $x^3 + x^2 + x + 1$. De este modo, se ve que la suma a nivel de bytes se corresponde claramente con una operación *EXOR*. En el Algoritmo Rijndael esta operación se realiza entre los bits de una clave o sub-clave y los correspondientes bits de una matriz de estado de resultados intermedios.

La multiplicación en *GF(2^8)* para la representación polinomial, se corresponde con la *multiplicación de polinomios módulo un polinomio binario irreducible de grado 8*. En *Rijndael* este polinomio se denomina *m(x)* y se presenta en la Ec.19:

$$m(x) = x^8 + x^4 + x^3 + x + 1 \qquad \text{(Ec.19)}$$

El polinomio *m(x)* suele expresarse en representación hexadecimal como *0x11B*. La multiplicación módulo *m(x)* dará siempre como resultado un polinomio de grado menor que *8* de tal manera que se pueda representar como un *byte*. Además cumple con la propiedad asociativa y posee como elemento neutro (*0x01*). Por ejemplo, la multiplicación entre los bytes (*0x57*) y (0x58) se puede expresar por medio de las siguientes ecuaciones:

Se puede observar que la suma es una operación mucho más sencilla a nivel de bytes que la multiplicación. Por otra parte, aplicando el *Algoritmo Extendido de Euclides* es factible encontrar para cada polinomio *b(x)* de grado menor que *8*, un polinomio *a(x)* y otro *c(x)* tal que se verifiquen las Ec.23 a Ec.25. De este modo se puede decir que existe el elemento inverso en la multiplicación.

$$b(x)a(x) + m(x)c(x) = 1 \qquad \text{(Ec.23)}$$

$$a(x).b(x) \bmod m(x) = 1 \qquad \text{(Ec.24)}$$

$$b^{-1}(x) = a(x) \bmod m(x) \qquad \text{(Ec.25)}$$

Existen entonces, *256* valores de bytes posibles y, si sobre ellos, se definen las operaciones suma y multiplicación mencionadas, el conjunto posee la estructura del campo finito *GF(*2^8*)*.

Puede pensarse la operación de multiplicación en un campo finito a partir de los elementos generadores de dicho campo. Las sucesivas potencias de estos elementos permiten generar progresivamente los *255* elementos diferentes de *0* de *GF(*2^8*)*. Cuando el valor del exponente llega a *255* se obtiene nuevamente el elemento unitario. Esta propiedad ofrece una manera de convertir la operación de multiplicación en una peración de suma.

Así, dos elementos a y b pertenecientes al campo pueden representarse por medio del elemento generador: a = g^{α} y b = g^{β}. Su producto $a.b$ podría expresarse como $a.b = g^{\alpha+\beta}$. Un elemento generador del campo *GF(*2^8*)* es el elemento (*0x03*)

Si se listaran en una tabla las diferentes potencias del elemento generador para cada elemento del campo finito, se podría conocer las potencias del generador, α y β, que originan los elementos a y b. La suma de ambos exponentes permitiría conocer la potencia del generador que corresponde al resultado de la multiplicación. Una nueva búsqueda en tabla permitiría hallar el resultado.

La consecuencia práctica más importante de la posibilidad de realizar la multiplicación por medio de una búsqueda en tabla es que dicha tabla también podría ser usada para hallar el elemento inverso de cada elemento del campo. Esto se debe a que el elemento g^x tiene como inverso al elemento g^{255-x}.

En el Algoritmo Rijndael estas propiedades se aplican a una transformación que los autores denominaron *SubBytes()* que posee las características de una sustitución no lineal . En los algoritmos criptográficos simétricos tipo bloque, este tipo de sustituciones suele llamarse *caja S*.

2.3.5.2.- Multiplicación de un Byte por *X*.

Si se multiplica el polinomio *x*, cuya representación hexadecimal es *0x02*, por otro polinomio *b(x)* considerando la representación polinomial de un *byte*, se obtiene:

$$b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \qquad \text{(Ec.26)}$$

El producto representado en la Ec.26 luego habrá que expresarlo en módulo *m(x)*. En el caso que b_7 sea *1* el cociente es *1* y el resto o resultado de la multiplicación se obtiene restando (por *suma módulo-2* ó *EXOR*) *m(x)* al dividendo. Si b_7 es *0* no hace falta dividir. De este modo, se deduce que la multiplicación por *x* se puede implementar a nivel de byte como un corrimiento a la izquierda seguido de una operación *EXOR* condicionada con *0x1B*.

La operación se denota como *b* = *xtime*(*a*) y en hardware dedicado sólo se implementaría mediante *4 EXORs*. La aplicación repetida de la misma sirve para efectuar multiplicaciones por potencias de *x*.

Por ejemplo, si se desea resolver el producto (0*x*57).(0*x*13) , como se verifica que (0*x*57).(0*x*13) = (0*x*57).(0*x*01⊕0*x*02⊕0*x*10) , se pueden realizan los siguientes cálculos:

$$(0x57)\cdot(0x02) = xtime(57) = AE \quad \text{(Ec.27)}$$

$$b_7 = 0 \Rightarrow leftshift \rightarrow 10101110 \quad \text{(Ec.28)}$$

$$(0x57)\cdot(0x04) = xtime(AE) = 47 \quad \text{(Ec.29)}$$

$$b_7 = 1 \Rightarrow leftshift, \oplus con1B \rightarrow 01011100 \oplus 0001011 = 01000111 \quad \text{(Ec.30)}$$

$$(0x57)\cdot(0x08) = xtime(47) = 8E \quad \text{(Ec.31)}$$

$$b_7 = 0 \Rightarrow leftshift \rightarrow 10001110 \quad \text{(Ec.32)}$$

$$(0x57)\cdot(0x10) = xtime(8E) = 07 \quad \text{(Ec.33)}$$

$$b_7 = 1 \Rightarrow leftshift, \oplus con1B \rightarrow 00011100 \oplus 00011011 = 00000111 \quad \text{(Ec.34)}$$

De este modo, la multiplicación entre los elementos del ejemplo resulta: 57.13=57*(01⊕02⊕10)=57⊕*AE*⊕07= *FE.* Así, la operación multiplicación por potencias de *x* puede generarse fácilmente mediante la aplicación repetida de la función *xtime()* y es particularmente útil en el caso en que dicha multiplicación se realice entre elementos tipo bytes y constantes. Sumando resultados parciales se puede generar de una manera sencilla la multiplicación por cualquier constante. Esta operación es usada en Rijndael y constituye un bloque básico para la implementación del algoritmo.

2.3.5.3.- Operaciones con Palabras de 32 bits.

Se pueden también definir polinomios de grado menor a *4* y con coeficientes en *GF(2^8)* para representar vectores de 4 bytes como lo indica la Ec.35.

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \qquad \text{(Ec.35)}$$

Esta representación es útil para aquellas operaciones de Rijndael que involucran palabras de 32 bits. Estos polinomios no son los mismos que los usados en la definición de elementos de campo finito, ya que los coeficientes en sí mismos son elementos de campo finito.

Se define la suma como la operación *EXOR* entre los respectivos coeficientes. En el caso de la multiplicación el resultado es un polinomio de mayor grado que no se puede representar por un vector de *4* bytes, por lo que hay que reducirlo mediante la operación módulo un polinomio de grado *4*. En Rijndael se utiliza para este propósito el polinomio de la Ec.36 y el mismo verifica la Ec.37.

$$M(x) = x^4 + 1 \qquad \text{(Ec.36)}$$

$$x^j \bmod x^4 + 1 = x^{j \bmod 4} \qquad \text{(Ec.37)}$$

Por otro lado, *M(x)* no es un polinomio irreducible sobre *GF(2^8)* por lo que la multiplicación por un polinomio fijo no es necesariamente invertible, pero en Rijndael el polinomio fijo elegido tiene inversa. Debido a la propiedad mencionada de *M(x)*, el producto *d(x) = a(x) □ b(x)* queda expresado como:

$$d(x) = (a_3x^3 + a_2x^2 + a_1x + a_0) \otimes (b_3x^3 + b_2x^2 + b_1x + b_0) \qquad \text{(Ec.38)}$$

$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0 \qquad \text{(Ec.39)}$$

donde

$$d_0 = a_0.b_0 \oplus a_3.b_1 \oplus a_2.b_2 \oplus a_1.b_3 \qquad \text{(Ec.40)}$$

$$d_1 = a_1.b_0 \oplus a_0.b_1 \oplus a_3.b_2 \oplus a_2.b_3 \qquad \text{(Ec.41)}$$

$$d_2 = a_2.b_0 \oplus a_1.b_1 \oplus a_0.b_2 \oplus a_3.b_3 \qquad \text{(Ec.42)}$$

$$d_3 = a_3.b_0 \oplus a_2.b_1 \oplus a_1.b_2 \oplus a_0.b_3 \qquad \text{(Ec.43)}$$

De este modo, se puede expresar la multiplicación "⊕" por un polinomio fijo *a*(x) como un producto de matrices:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad \text{(Ec.44)}$$

2.3.5.4.- Multiplicación de una Palabra de 32 bits por X.

Un caso especial lo constituye la multiplicación $x \oplus b(x)$. El resultado es un polinomio de grado *4*, $b_3x^4+b_2x^3+b_1x^2+b_0x$, que debe reducirse módulo *M(x)*. Al realizar la reducción el polinomio resultante adopta la forma $b_2x^3+b_1x^2+b_0x+b_3$. Si se piensa la operación desde el punto de vista matricial, es equivalente a la multiplicación por una matriz cuyos a_j=00 excepto a_l= 01, como se expresa en la Ec.45.

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 00 & 00 & 00 & 01 \\ 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad \text{(Ec.45)}$$

De este modo, la multiplicación por x se convierte en un corrimiento cíclico de los bytes dentro del vector en cuestión. La función *RotWord()* que se utiliza en la generación de las diversas sub-claves para cada ronda corresponde a una multiplicación por x^3.

CAPITULO III

3.- TRABAJOS SIMILARES.

A continuación se presentan algunos sistemas que han implementado encriptación y compresión de archivos electrónicos en hardware:

3.1.- ARQUITECTURA DEL HARDWARE DE UN CRIPTOSISTEMA DE CURVA ELÍPTICA CON COMPRESIÓN DE DATOS.

Este sistema presenta el diseño e implementación de una arquitectura hardware de un sistema de cifrado/descifrado de datos mediante criptografía de curva elíptica que opera en tiempo real. Los datos a cifrarse se comprimen primero de manera transparente mediante un algoritmo de compresión sin pérdida también implementado en hardware. Una de las ventajas de este enfoque es el poder mejorar el rendimiento del sistema al reducir la cantidad de información a cifrarse, además de beneficiarse de transmitir más información con el mismo ancho de banda disponible.

3.2.- IMPLEMENTACIÓN EN UN FPGA DEL MODELO DE COMPRESIÓN DE DATOS PPMC.

La implementación en hardware de este tipo de algoritmos se realizó, un diseño minucioso que maximiza su rendimiento y demanda pocos recursos, usando estructuras eficientes para almacenar datos. Entre estas estructuras están los arreglos CAM (Content-Addressable Memory) que permiten la búsqueda simultánea en todas las entradas del diccionario.

El modelo de compresión implementó el algoritmo PPMC de orden 0, por lo que se cuenta con dos diccionarios, uno para contextos de orden 0 y otro para orden -1.

3.3.- IMPLEMENTACIÓN DEL ALGORITMO CRIPTOGRÁFICO IDEA EN VIRTEX USANDO JBITS.

En este trabajo se describe una implementación desarrollada con JBits del algoritmo criptográfico IDEA, utilizando una FPGA de la familia Virtex. Se ha realizado un diseño por constantes del multiplicador módulo y del sumador para reducir significativamente el área.

JBits como herramienta de diseño permite unificar bajo un mismo lenguaje la descripción de circuitos y los programas. De esta manera, en la aplicación propuesta la generación de las subclaves se ha realizado por la parte software.

3.4.- IMPLEMENTACIÓN EN HARDWARE DEL ALGORITMO SERPENT.

El algoritmo Serpent es un algoritmo para criptografía simétrica, el cual emplea 32 vueltas. La longitud del bloque de texto es de 128 bits, y la longitud de la clave puede ser de 128, 192 o 256 bits.

Este artículo presenta el diseño de tres arquitecturas en hardware para el algoritmo Serpent. En este caso, las arquitecturas son iterativas, y fueron diseñadas considerando las ventajas que presenta el hardware para implementar algunas transformaciones del algoritmo Serpent con respecto a las implementaciones en software.

CAPITULO IV

4.- DASARROLLO

4.1.- INTERFAZ DE INGRESO DE DATOS

La interfaz entre el usuario y el FPGA, en este trabajo, fue llevada a cabo con la ayuda del software LabVIEW 8.6, de National Intruments, por contar con la paquetería para la configuración del FPGA que se encuentra en el kit de aprendizaje de la tarjeta Spartan 3E, tomando en cuenta que la comunicación de la tarjeta es vía USB, por lo que conlleva a disponer de diversas configuraciones como se muestra en la *tabla x1*, en nuestro caso utilizaremos, para la comunicación US, la opción de la Master serial (M.S).

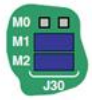
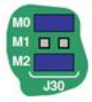
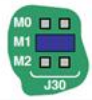
Configuration Mode	Mode Pins M2:M1:M0	FPGA Configuration Image Source	Jumper Settings
Master Serial	0:0:0	Platform Flash PROM	
SPI (see Chapter 12, "SPI Serial Flash")	1:1:0	SPI Serial Flash PROM starting at address 0	
BPI Up (see Chapter 11, "Intel StrataFlash Parallel NOR Flash PROM")	0:1:0	StrataFlash parallel Flash PROM, starting at address 0 and incrementing through address space. The CPLD controls address lines A[24:20] during BPI configuration.	
BPI Down (see Chapter 11, "Intel StrataFlash Parallel NOR Flash PROM")	0:1:1	StrataFlash parallel Flash PROM, starting at address 0x1FF_FFFF and decrementing through address space. The CPLD controls address lines A[24:20] during BPI configuration.	
JTAG	0:1:0	Downloaded from host via USB-JTAG port	

Tabla No 15: *Opciones de configuración para la spartan 3E*

Realizamos diversas pruebas para trabajar con el monitor led *done* (*denominado por el Manual de Usuario de la S-3E),* que es un indicador, el cual nos informa cuando el programa o la configuración es exitosa, como se muestra en la figura Nº 4.

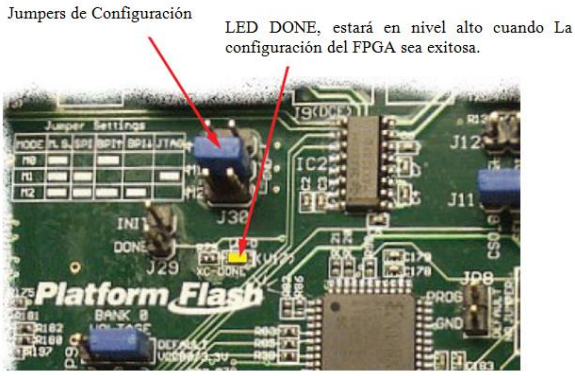


Figura No. 4: *Ubicación de jumpers de configuración y led done*

Con la ayuda del programa, ya antes mencionado LabVIEW 8.6, fue posible realizar la interfaz grafica del software, debido a que el entorno para el usuario es más sencillo de manejar, este diseño tiene la característica de que por sí solo arregla la manera de adquirir datos desde un teclado PS/2, los cuales son procesados por el FPGA, para implementar el algoritmo de encriptación al ser mostrados en el LCD si el valor arrojado por éste es un carácter; y en caso de no mostrar un carácter obtenemos el valor binario de la encriptación en los indicadores del programa y el FPGA.

4.1.1.- Interfaz Grafica en LABVIEW 8.6

LabVIEW es una herramienta gráfica que usa lenguaje G, donde la G simboliza que es lenguaje Gráfico, esté es muy importante para la adquisición de datos hacia el FPGA de la tarjeta Spartan 3E, ya que facilita al programador las configuraciones del FPGA para poder acceder a los diversos puertos con los que cuenta.

En esta etapa se realizó la interfaz gráfica para poder ver los datos que están siendo procesados por el FPGA, así mismo como los resultados obtenidos de las operaciones que se realizan en la encriptación y des-encriptación de datos.

Para el mejor desarrollo del programa se utilizó la herramienta *"tab control"*, la cual permite dividir en pestañas las aplicaciones de encriptación y des-encriptación como se muestra en la *figura No.5*, esta es una forma más práctica de tener los dos procesos en una sola aplicación para que el usuario no tenga complicaciones.

Figura No. 5: *Localización herramienta Tab control*

En el programa se realizaron diversos archivos SubVis para poder hacer más fácil la comprensión del programa como las debidas configuraciones para inicializar y configurar el LCD y teclado PS/2.

La interfaz gráfica cuenta con dos pestañas, en la primera nos muestra la ventana en donde a través de unas etiquetas, accederemos los caracteres para la encriptación de datos, y otra donde se arrojara el resultado de la encriptación, además de un contador, cuando tenemos un carácter en el LCD, y un indicador binario como se muestra en la *figura No 6.*

*E*n la segunda pestaña nos presenta la interfaz para la des-encriptación de datos, la cual tiene dos etiquetas que nos muestra el valor recibido y el resultado en hexadecimal, como se muestra en la *figura No. 7.*

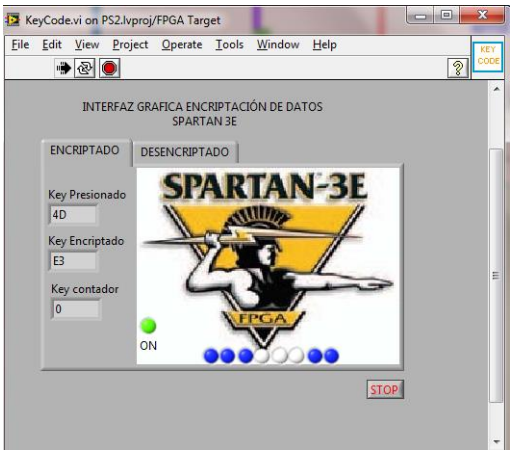


Figura No. 6: *Interfaz grafica de encriptado*

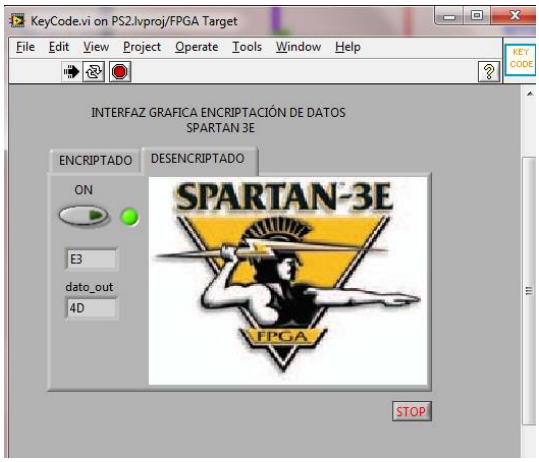


Figura No. 7: *Interfaz grafica des-encriptado*

Los valores mostrados provienen del teclado PS/2, los cuales ya son predeterminados del mismo, y solamente se realizó la configuración de las etiquetas para que pudiesen mostrar los datos en valores hexadecimales, como se puede ver en la *figura No. 8*.

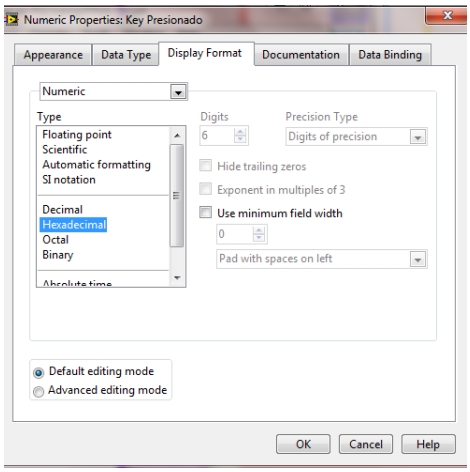


Figura No. 8: *Propiedades de las etiquetas en labview*

4.2.- Adquisición de datos en la Spartan 3E con teclado PS/2

Esta tarjeta es una alternativa para el diseño y evaluación de sistemas. Una de las características principales con las que cuenta son los puertos de adquisición de datos, en nuestro caso haremos uso del puerto PS/2, además los Slide switches.

Al crear el proyecto en LabVIEW se especificó la tarjeta con la que trabajamos, al ir a la ventana de "*Add Targets and Devices*", ahí seleccionamos la opción de "*New target or device*" por lo que reconocerá automáticamente la tarjeta Spartan 3E, ya que se cuenta con el complemento ya instalado del FPGA para LabVIEW 8.6, como se muestra en la *figura No. 9*.

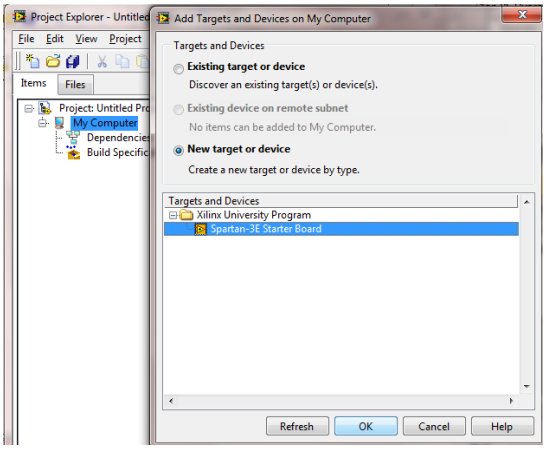


Figura No. 9: *Ventana de "Add Targets and Devices"*

Los elementos y puertos que usamos en la tarjeta, fueron agregados gracias a la opción de "*FPGA I/O*", en ésta se encuentran todos los puertos disponibles para la tarjeta, así se tuvo acceso a ellos y se configuraron de forma gráfica, con las direcciones expresadas en bloques como se muestra en la *figura 10*.

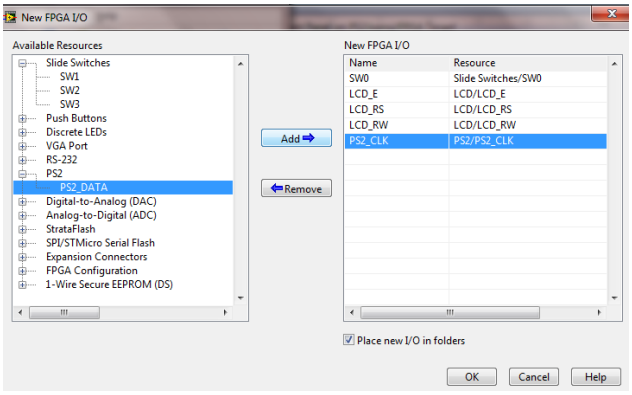


Figura No. 10: *Ventana de "FPGA I/O"*

4.2.1 Inicialización y Configuración del teclado por puerto PS/2

La configuración que se realizó en LabVIEW para la recepción de datos mediante el teclado hacia la tarjeta Spartan 3E, fue prescindiendo de los pines PS2_DATA y PS2_CLK del puerto PS/2 de la Spartan 3E, la ubicación del puerto y de los pines se indica en la *figura 11*.

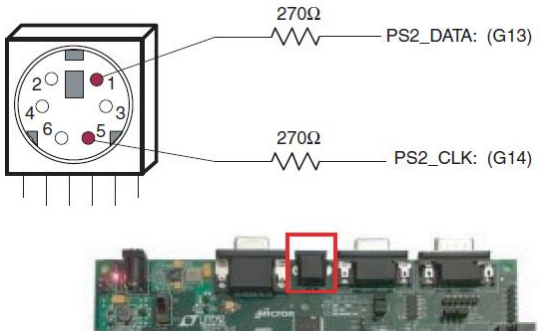


Figura No. 11: *Localización y pines del puerto PS/2*

El puerto PS/2 tiene una configuración estándar en la tarjeta spartan 3E y solamente los pines 1, 3, 4 y 5, son necesarios para la comunicación con la tarjeta como se muestra en la *tabla 16.*

PS/2 DIN Pin	Signal	FPGA Pin
1	DATA (PS2_DATA)	G13
2	Reserved	G13
3	GND	GND
4	+5V	—
5	CLK (PS2_CLK)	G14
6	Reserved	G13

Tabla No. 16: *Pines del puerto PS/2*

El teclado PS/2-style puede usar códigos para comunicarse con datos de teclas presionadas. Casi todos los teclados usados hoy en día son del tipo PS/2. Cada tecla es singular, con un código único, cuando la tecla es presionada en cualquier momento esta realiza un escaneo para enviar el dato. El código de la mayoría de las teclas se muestra en la *figura x9.*

Cuando la tecla está siendo presionada o sostenida, el escaneado se hace repetidamente para detectar que se está enviando, la velocidad del escaneado es de 100 ms. Si la tecla se suelta, el teclado envía un código "F0" que es expresado como *tecla levantada* (key-up).

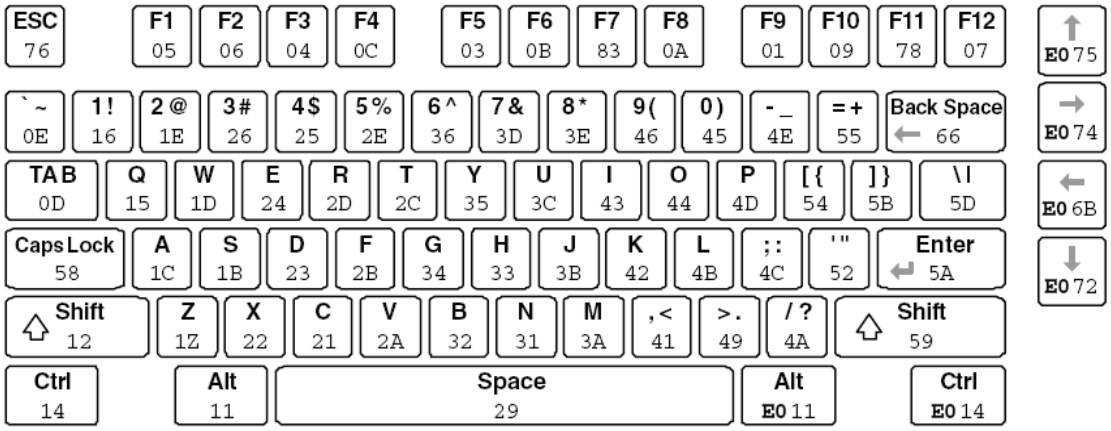


Figura No. 12 : *Valores del teclado PS/2*

El teclado envía datos al puerto PS/2 de la tarjeta en paquetes de 11-bits, que contienen a '0' bit de inicio, seguido por códigos de paquetes de 8-bits de escaneo (el primero es el bit menos significativo), seguido por un bit de paridad impar y terminado con un '1' que es el bit de paro. Cuando el teclado envía datos, estos generan 11 transiciones de reloj de 20 a 30 Khz, y el dato es validado en el flanco de caída del pulso de reloj como se muestra en la *figura 13.*

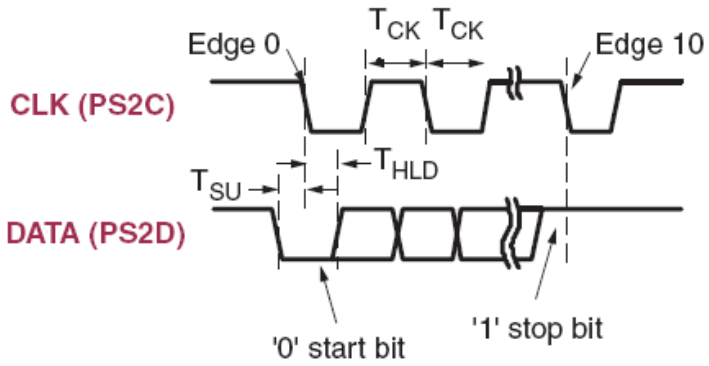


Figura No. 13: *Graficas de tiempo del teclado*

Para la configuración en LabVIEW del puerto PS/2, se agregaron desde la opción *FPGA I/O* los pines PS2_DATA y PS2_CLK, para tener acceso a ellos en el diagrama de bloques de LabVIEW y tener comunicación con el FPGA, para tener una lectura continua de datos se agregaron los bloques de estos dos pines dentro de un ciclo while loop, para tener el control de cuando leer datos del puerto PS/2 de la tarjeta Spartan 3E y finalizar el programa; como se muestra en la siguiente *figura 12*.

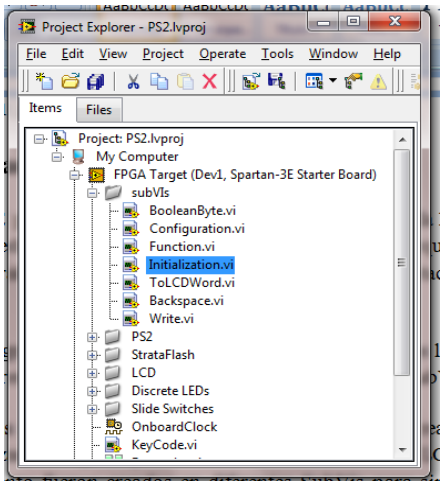


Figura No. 14: *Bloques para el control del puerto PS/2*

4.3.- Envió de datos para el LCD y LEDS

La tarjeta Spartan 3E posee dentro de su hardware, una pantalla LCD de 16 columnas por 2 líneas y 8 Leds como monitores de salida. Estos indicadores son de vital importancia, ya que con ellos podremos visualizar que trabajo se está realizando, ya sea el programa de encriptación o el de des-encriptación.

La programación grafica de LabVIEW nos facilita tener acceso a los leds, debido a que estos solamente reciben valores booleanos.

El LCD requiere de configuraciones más avanzadas, que fueron realizadas en LabVIEW, como el de inicializar. Configurar y describir el funcionamiento del LCD, para la programación de cada elemento fueron creados en diferentes SubVis para simplificar el programa principal como se muestra en la *figura No.15*.

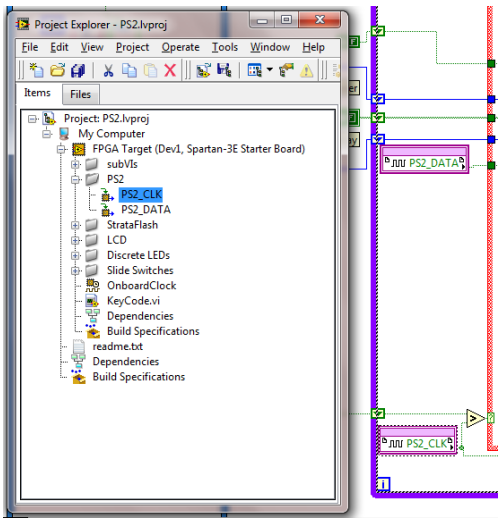


Figura No.15 Configuración y descripción del funcionamiento del LCD con Sub Vis

4.3.1.- Inicialización y Configuración del LCD.

La tarjeta Spartan 3E controla al LCD a través de una interfaz de datos de 4-bits para la compatibilidad con otros dispositivos y tarjetas de Xilinx, logrando de esta manera minimizar el número total de pines, como muestra la *figura no 16*. Además el LCD soporta una interface de datos de 8-bits.

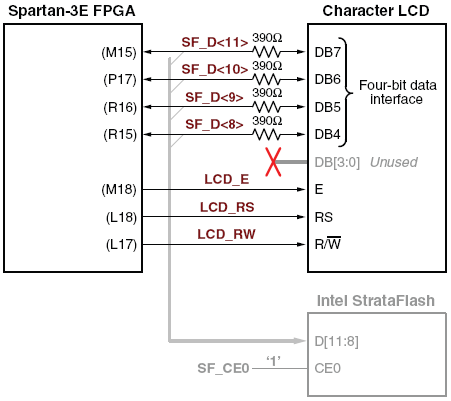


Figura No. 16: *Interfaz de carácter para el LCD*

El LCD es una manera práctica de monitorear una gran variedad de información usando el código estándar ASCII, que cuenta con caracteres ya predeterminados. Sin embargo, este Display no es muy rápido.

El despliegue de la pantalla sucede en intervalos de medio segundo. Comparado con el reloj de 50Mhz disponible en la tarjeta, el display es lento. El LCD tiene configuraciones que se tienen que realizar para que esta funcione correctamente como se muestra en la *tabla 17*.

Signal Name	FPGA Pin	Function	
SF_D<11>	M15	Data bit DB7	Shared with StrataFlash pins SF_D<11:8>
SF_D<10>	P17	Data bit DB6	
SF_D<9>	R16	Data bit DB5	
SF_D<8>	R15	Data bit DB4	
LCD_E	M18	Read/Write Enable Pulse 0: Disabled 1: Read/Write operation enabled	
LCD_RS	L18	Register Select 0: Instruction register during write operations. Busy Flash during read operations 1: Data for read or write operations	
LCD_RW	L17	Read/Write Control 0: WRITE, LCD accepts data 1: READ, LCD presents data	

Tabla No. 17: *Tabla de configuración para la interfaz del LCD*

Las cuatro señales de datos del LCD son también compartidos con las líneas del StrataFlash SF_D<11:8>. Como se puede ver en la *Tabla x4*, la interacción del LCD/StrataFlash depende de las aplicaciones usadas en el diseño. Cuando la memoria del StrataFlash esta deshabilitada (SF_CE0 =alto), las aplicaciones del FPGA están listas para acceder a la lectura/escritura del LCD. De manera inversa, cuando en el LCD esta deshabilitada la operación de lectura (LCD_RW = bajo), la aplicación de lectura/escritura del FPGA está lista para acceder a la memoria del StrataFlash.

SF_CE0	SF_BYTE	LCD_RW	Operation
1	X	X	StrataFlash disabled. Full read/write access to LCD.
X	X	0	LCD write access only. Full access to StrataFlash.
X	0	X	StrataFlash in byte-wide (x8) mode. Upper address lines are not used. Full access to both LCD and StrataFlash.

Tabla No 18: *Control de interacción entre el LCD y la memoria StrataFlash*

El controlador tiene internamente tres regiones de memoria, cada una con un propósito en específico. El display debe de ser inicializado antes de acceder a cualquiera de estas regiones de memoria.

El Display Data RAM (DD RAM) almacena el código de caracteres que se visualizan en la pantalla. La mayoría de las aplicaciones interactúan primeramente con la DD RAM. Los caracteres almacenados en la DD RAM tienen una locación de referencia en un mapa de bits, también un carácter *set* predeterminado en una CG RAM o el uso-definido carácter SET CG RAM.

Las direcciones que vienen predeterminadas por los 32 caracteres localizados en el display. La primera línea de caracteres es guardado entre las direcciones 0x00 y 0x0F. La segunda línea de caracteres es localiza entre las direcciones 0x40 y 0x4F como se muestra en la *figura x14*.

Dirección de los Caracteres del Display

																Undisplayed Addresses		
1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	...	27
2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	...	67
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...	40

Tabla No. 18 : *Direcciones en hexadecimal para la DD RAM*

La pantalla está fija para una operación 4-bit, cada comando de 8-bit es enviado en un par de paquetes de 4-bits. El paquete más significativo es el primero, y el paquete que sigue es el menos significativo, en el SubVis que se creó para realizar esta función se muestra en la *figura 17.*

Figura No 17: *Configuración para la escritura de datos en el LCD*

Para realizar estas operaciones se configuraron funciones las cuales son:

Habilitación: Si la señal habilitada LCD_E es de nivel bajo, todos los demás entradas del LCD serán ignorados.

Limpiar pantalla: Para limpiar la pantalla y regresar el cursor a la posición inicial, a la esquina superior izquierda. Este comando escribe un espacio en blanco (ASCII/ANSI el código del carácter es 0x20) dentro de todas las direcciones de la DD RAM.

En LabVIEW estas configuraciones quedan expresadas como muestra la *figura 18.*

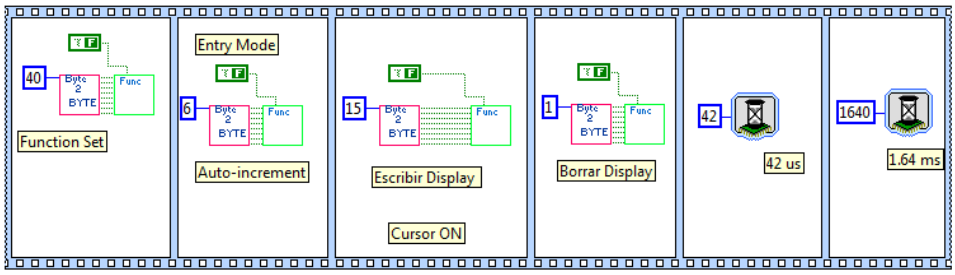


Figura No. 18: *SubVI para configurar el Display*

La pantalla debe ser inicializada para establecer la comunicación como lo requiere el protocolo. Para iniciar la secuencia de inicialización primero establece que la aplicación del FPGA desea usar la interface de 4-Bit de dato al LCD como se muestra en la *figura 19* y en los pasos siguientes:

1. Espera 15 ms o más, Sin embargo la pantalla generalmente esta lista cuando el FPGA termina la configuración. Los 15ms se dividen en intervalos de 750,000 ciclos de reloj a 50 MHz.
2. Escribe SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock cycles.
3. Espera 4.1 ms o más, cada 205,000 ciclos de reloj a 50 MHz.
4. Escribe SF_D<11:8> = 0x3, pulsa LCD_E en nivel alto para 12 ciclos de reloj.
5. Espera 100 µs o más, que es de 5,000 ciclos de reloj a 50 MHz.
6. Escribe SF_D<11:8> = 0x3, pulsa LCD_E en nivel alto para 12 ciclos de reloj.
7. Espera 40 µs o más, que es 2000 ciclos de reloj a 50 MHz.
8. Escribe SF_D<11:8> = 0x2, pulsa LCD_E en nivel alto parar 12 ciclos de reloj.
9. Espera 40 µs o más, que es 2000 ciclos de reloj a 50 MHz

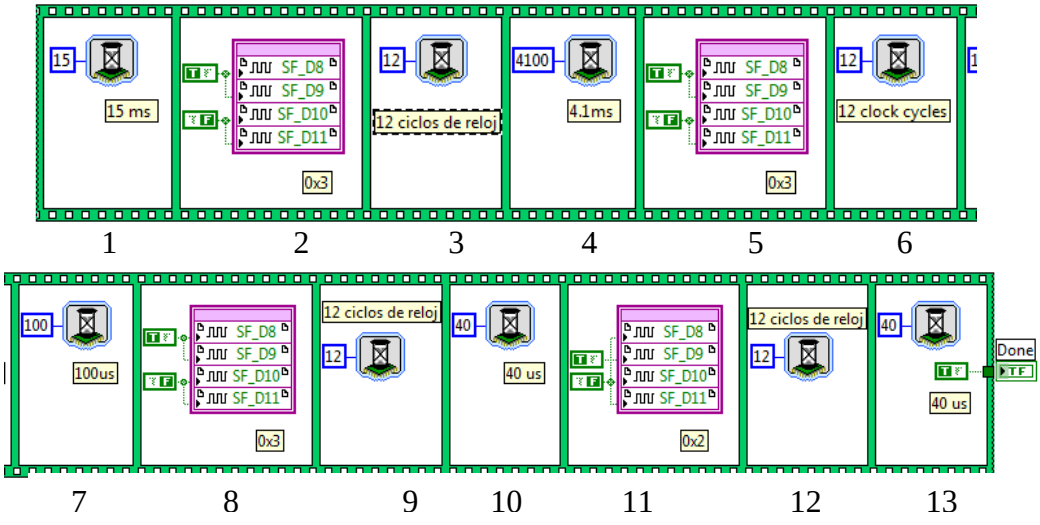


Figura No. 19: *SubVI para inicializar el Display*

4.4.- IMPLEMENTACIÓN DEL ALGORITMO RIJNDAEL EN SPARTAN 3E

El Algoritmo Rijndael es un cifrador de bloque con una arquitectura interna con realimentación, en inglés *looping*, en la que los datos pasan iterativamente a través de una función denominada ronda. Existen varias opciones en cuanto a la implementación de este tipo de arquitectura que pueden conducir a diseños optimizados.

En el caso en que un algoritmo presenta una estructura interna homogénea en cada ronda de su estructura iterativa, probablemente pueda implementarse en hardware como una estructura de ronda en lógica combinacional. Es decir su estructura constará de circuitos cuyas salidas dependan solamente de las entradas presentes. En un período de reloj del sistema los datos ingresarán al circuito a través de algún elemento multiplexor y la salida resultante se almacenará en algún registro. En cada período de reloj subsiguiente se evaluaría una ronda del algoritmo, precisándose para el arquitectura suele utilizarse en modos criptográficos con realimentación para aumentar los parámetros de eficiencia relacionados con la velocidad del circuito.

Para la implementación en la tarjeta Spartan 3E se realizaron dos operaciones del algoritmo rijndael la operación *SubBytes y mixcolum.*

4.4.1.- FUNCIÓN SubBytes

La función SubBytes consiste en una sustitución sobre cada *byte* del bloque de entrada o Matriz de Estado *s*. La posición del *byte* en dicha matriz no es un aspecto relevante y es independiente, en cuanto a su transformación, de la aplicación de ésta operación sobre los demás *bytes*. Matemáticamente, la operación consiste de dos pasos:

1. Inversión del byte en *GF* (2^8). En este paso el *byte* en cuestión se toma como un elemento del campo.

2. Aplicación de la transformación afín. En este paso la representación más apropiada para un *byte* es la de un vector de 8 bits como se muestra en la siguiente Ec.46:

$$x \rightarrow Ax \oplus b$$

Con

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \qquad \textbf{Ec. 46}$$

$$b = (1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0)^T$$

La diferencia en la representación en ambos casos genera la relación de alinealidad que se busca que presente cualquier caja S aplicable a un algoritmo criptográfico. Vale la pena recalcar que la operación equivalente de des encriptado significa trabajar con la matriz inversa de *A*, *A-1*, y un vector *c*, diferente del *b* pero obtenido a partir de él.

Para realizar la transformación de SubByte, la caja S fue capturada en una herramienta de LabVIEW "Hdl interface node", la cual nos proporciona la facilidad de poder programar en VHDL, poder declarar tanto entradas como salidas y sea procesada por el FPGA, de forma rápida, la *figura x18* nos muestra un ejemplo de cómo queda el *hdl interface node* después de ser programado.

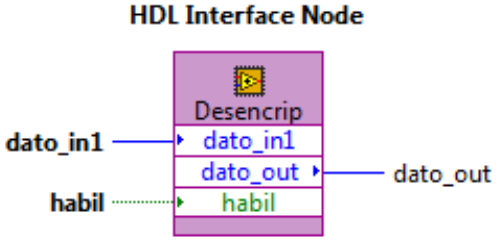


Figura No. 19: *Diagrama del HDL interface node*

4.4.2.- FUNCIÓN MixColumn

La función MixColumn esta operación consiste en una multiplicación de polinomios sobre GF (2^8), módulo $x^4 + 1$. Uno de los polinomios de la multiplicación se toma a partir de una columna de *State* y el otro es un polinomio fijo *a(x).*Esta multiplicación tiene una representación matricial. Dicha ecuación puede re-escribirse de la siguiente forma:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} s_{0,c} + \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} s_{1,c} + \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} s_{2,c} + \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} s_{3,c}$$

Ec.47

Es de notar que las columnas de la Ec.47 sólo presentan tres constantes: 01, 02 y 03, lo cual simplifica la operación pues se traduce en la multiplicación por potencias de x de bajo exponente.

A su vez, la Ec.48 puede re-escribirse como cuatro ecuaciones:

$$s'_{0,c} = (0x02) \cdot s_{0,c} \oplus (0x03) \cdot s_{1,c} \oplus s_{2,c} \oplus s_{3,c} \qquad \textbf{Ec. 48}$$

$$s'_{1,c} = s_{0,c} \oplus (0x02) \cdot s_{1,c} \oplus (0x03) \cdot s_{2,c} \oplus s_{3,c} \qquad \textbf{Ec. 49}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (0x02) \cdot s_{2,c} \oplus (0x03) \cdot s_{3,c} \qquad \textbf{Ec. 50}$$

$$s'_{3,c} = (0x03) \cdot s_{0,c} \oplus s_{1,c} \oplus s_{2,c} \oplus (0x02) \cdot s_{3,c} \qquad \textbf{Ec. 51}$$

La presencia de sólo dos constantes significativas en las multiplicaciones, *02* y *03*, permite aprovechar las propiedades mencionadas a la operación *xtime ( )*. Desde el punto de vista de implementación en hardware, la operación mencionada se puede imaginar como el circuito representado en la *figura x19*.

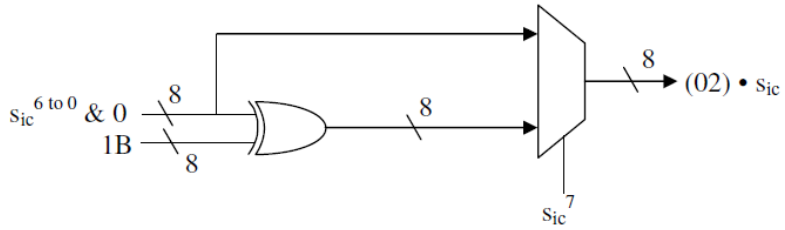


Figura No.20: *Circuito multiplicador por (02). Funcionalidad*

Una compuerta EXOR realiza esta operación entre dos entradas: un byte de datos desplazado a la izquierda y un byte fijo, en hexadecimal la constante 0x1B. El resultado de la multiplicación por (0x02) puede ser el byte de datos desplazado o la salida de la EXOR, según el valor del bit más significativo, s_{ic} 7, del byte de datos en cuestión. De esta manera con una función EXOR y un multiplexor de doble entrada se puede generar la operación xtime (). En líneas de código VHDL esto se traduce en una arquitectura muy sencilla.

El multiplicador mencionado sirve de base para la implementación de un multiplicador por *(0x03)*, como se representa en la *figura x20*. El componente surge de la descomposición de esta constante en la suma 0*x*01 exor 0*x*02 = 0*x*03 y la aplicación de la propiedad distributiva de la multiplicación respecto de la suma.

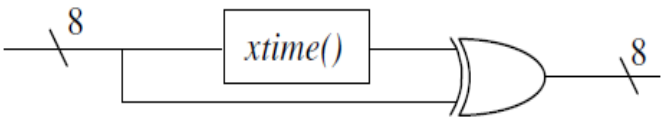


Figura No. 21: *Circuito multiplicador*

4.4.3.- CLAVE DEL SISTEMA DE ENCRIPTACIÓN

La clave del sistema es ingresada por el usuario en la programación vhdl que se realiza en la herramienta de Labview *"Hdl interface node",* esta función es muy sencilla de realizar ya que solo se implemento una compuerta *EXOR* con dos buses de entrada de *8 bits,* es una función muy sencilla de realizar en *VHDL*.

La clave del sistema fue declarada como una señal dentro de la programación y pueda ser modificada por el usuario.

CAPITULO IV

5.- DISEÑO E IMPLEMENTACIÓN

5.1.- RECOPILACIÓN DE INFORMACIÓN.

Se recopiló información y conocimientos de algoritmos de compresión y encriptación de archivos electrónicos implementados en "Hardware". Se realizaron búsquedas en Internet, para obtener información más globalizada y actualizada. De esta manera fue posible recabar información y observaciones de los diferentes algoritmos de compresión y encriptación de archivos electrónicos y su funcionamiento dentro de un sistema. Se realizaron verificaciones de trabajos similares, apoyado en los registros e información generada en residencias anteriores y trabajos profesionales de tesis sobre el mismo tema.

5.2 VERIFICACIÓN Y ANÁLISIS DE ALGORITMOS.

Verificación y análisis del funcionamiento de los diferentes algoritmos de compresión y criptográficos, fue una tarea un poco complicada, pues la información es basta. La información recabada fue organizada, para establecer los parámetros y condiciones en que se utilizarán para seleccionar el algoritmo de compresión y encriptación de archivos electrónicos a implementar en el circuito digital. Se revisaron algunos algoritmos, pero se selecciono el que presentaba una implementación factible.

5.3 DISEÑO DEL SISTEMA DE ENCRIPTACIÓN Y COMPRESIÓN

El sistema cuenta con programa desarrollado en Visual Basic 6.0, el cual es la interfaz grafica que nos permite comprimir y encriptar por hardware a un archivo de texto plano, a continuación se muestra un diagrama a bloques del sistema:

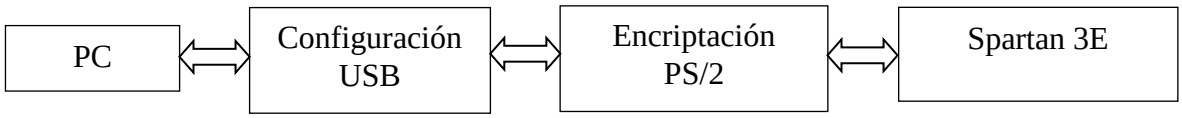


Figura No. 22 *"Diagrama a bloques del sistema"*

Para el diseño del sistema de encriptación y compresión, se selecciono algunos algoritmos más adecuados que permitieran su implementación en un microcontrolador.

A continuación se presenta una explicación detallada de la forma en que se adaptaron los algoritmos seleccionados para la compresión y Encriptación de texto plano.

5.3.1 Compresión de archivos de texto plano

5.3.1.1 Compresión

Para realizar esta actividad se analizó el algoritmo de Huffman muy detalladamente y se tomo la decisión de implementar y retomar la idea de codificar las letras a un número de bits inferior a un byte. La idea de esta codificación es codificar a 4 bits cada carácter, para que de esta forma por cada dos caracteres haya un ahorro de un carácter, obteniendo una compresión del 50 % del texto plano, a continuación se explica en que consiste dicha codificación:

1.-Generar un matriz de 14x14, es decir, 196 elementos los cuales son caracteres que pertenecen al ASCII extendido. La comunicación serial que implementamos solo nos permite enviar un byte, es decir, un carácter perteneciente al ASCII.

Los caracteres ASCII, que van del 0 al 31 se llaman de comandos, y sirven para realizar operaciones como el retorno de carro, timbre, etc. De esto deducimos que tenemos que utilizar los caracteres imprimibles, es decir, del 32 al 255. Asignamos valores a partir del valor 32, el 32 como elemento cero de la matriz, 33 como el segundo elemento, 34 como tercer elemento, así sucesivamente hasta llegar a la columna 14, luego cambiamos de fila a la fila 1, y comenzamos a llenar nuevamente con el valor siguiente, realizamos este proceso hasta llenar toda la matriz.

	a	**c**	**d**	**e**	**I**	**l**	**m**	**n**	**O**	**p**	**r**	**s**	**t**	**u**
A		!	“	#	$	%	&	‘	(	)	*	+	,	-
C	.	/	0	1	2	3	4	5	6	7	8	9	:	<
D	=	>	¿	@	A	B	C	D	E	F	G	H	I	J
E	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
I	Y	Z	[	\	]	^	_	`	A	b	c	d	e	f
L	g	h	i	j	K	l	m	n	O	p	q	r	s	t
M	u	v	w	x	Y	z	{	|	}	~	⌂	Ç	ü	é
N	â	ä	à	å	Ç	ê	ë	è	Ï	î	ì	Ä	Å	É
O	æ	Æ	ô	ö	Ò	û	ù	ÿ	Ö	Ü	ø	£	Ø	×
P	*ƒ*	á	í	ó	Ú	ñ	Ñ	ª	º	¿	®	¬	½	¼
R	¡	«	»	░	▒	▓	│	┤	Á	Â	À	©	╣	║
S	╗	╝	¢	¥	┐	└	┴	┬	├	─	┼	ã	Ã	╚
T	╔	╩	╦	╠	═	╬	¤	ð	Ð	Ê	Ë	È	ı	Í
U	Î	Ï	┘	┌	█	▄	¦	Ì	▀	Ó	ß	Ô	Ò	õ

Tabla No.19 *“Matriz con alfabeto”*

2.- Codificar las 14 letras más usadas en el español, las cuales son:

acdeilmnoprstu

La letra “a” corresponde a la fila cero, la letra “c” a la fila uno, la letra “u” corresponde a la fila 13. A las columnas se les asigna el mismo código que el de las filas mencionado anteriormente.

3.- Tomar en pares las letras del texto plano a comprimir, recordar que deben de estar comprendidas entre las 14 letras mencionadas anteriormente. Se toma la primer letra y se revisa que letra es de las 14, conocida la letra se le asigna el código correspondiente del numero 0 al 13, dicho numero corresponde a la fila de la matriz, y para la segunda letra se realiza la misma operación de revisar que letra es, y después se hace corresponder ese valor a la columna de la matriz, de este modo con las dos letras que se toman se apunta a un elemento, cuya ubicación se encuentran determinada por las dos letras mencionadas. El elemento que nos regresa la matriz es un carácter que contiene la información de dos caracteres, al realizar esta actividad estamos comprimiendo un texto plano a la mitad de su tamaño.

Carácter	Código Asignado
A	0
C	1
D	2
E	3
I	4
L	5
M	6
N	7
O	8
P	9
R	10
S	11
T	12
U	13

Tabla No. 20 *"Código de asignación"*

Ejemplo:

Si tomamos el texto plano: "paco", se identifica a la primera letra y se le asigna el valor correspondiente según la tabla anterior, luego tomamos la segunda letra, se identifica, y después la codificamos con el número de la tabla, realizando esta actividad dos veces, para este ejemplo el resultado es el siguiente:

Pares de letras del texto	Fila	Columna	Elemento
pa	p=9	a=0	***f***
co	c=1	o=8	**6**

Tabla No. 21: *"Compresión de texto plano"*

La asignación de un número a cada uno de los caracteres es importante, para tener un control total en el momento de apuntar el elemento en la matriz, es decir, para especificar la fila y columna correspondiente.

Como se observó en la tabla anterior la palabra "paco" de cuatro bytes, después de aplicarle esta codificación a 4 bits, nos entrega únicamente dos caracteres obteniendo como resultado una compresión del 50% de la palabra original.

5.3.1.2 Descompresión de archivos de texto plano

Los pasos necesarios para llevar acabo esta sección se mencionan a continuación:

1.-Se toma el primer elemento a descomprimir,
2.-luego se realiza una búsqueda a través de la matriza de 14x14 hasta encontrar la fila y columna en la que se encuentra este símbolo.
3.-Conociendo la ubicación del elemento, conocemos cuales son las dos letras que se comprimieron, tomamos como el primer carácter al valor obtenido por la fila, el segundo carácter es el que ocupa el valor de la columna. Para visualizar mejor este proceso revisemos el texto compreso anteriormente, el texto compreso es *f6.*

Texto comprimido	Fila	Columna	Texto descomprimido
f	**p**	**a**	**pa**
6	**c**	**o**	**co**

Tabla No. 22: *"Descompresión de texto plano"*

El texto descomprimido que finalmente se obtiene es **paco.**

Se tomo la decisión de implementar el compresor de archivos de texto utilizando un microcontrolador de la familia 18F, de Microchip, debido al tamaño de su memoria ROM y RAM.

CAPITULO V

7.- RESULTADOS

Los resultados planteados en este capítulo se basan en la adquisición de datos desde el puerto PS/2 para la encriptación y des encriptación de datos, en un FPGA con una interfaz grafica en LabVIEW usando la tarjeta Spartan 3E, el resultado se muestra en la *figura x21*.

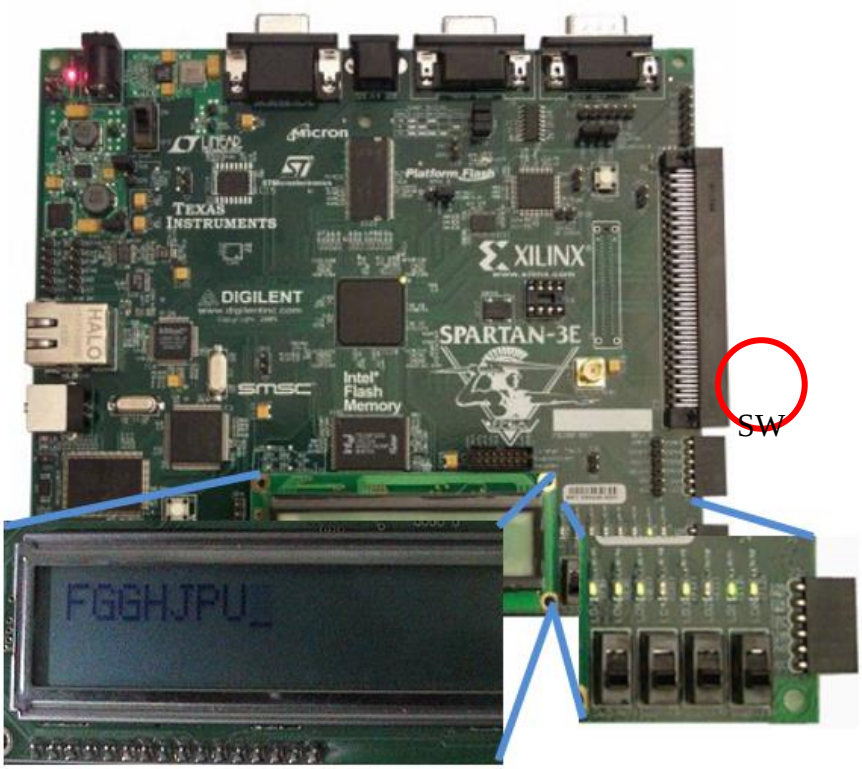


Figura No. 23: *Tarjeta Spartan 3E, muestra el resultado de la encriptación y des encriptación*

De la implementación y de la unión de las diferentes etapas que se desarrollaron para la encriptación y des encriptación de datos, se obtuvo el siguiente resultado, utilizando las funciones del capítulo 3, como se muestra en la *figura no. 23*.

El ejemplo mostrado en la *figura no. 23* es el valor hexadecimal de la letra U el cual es '3C', al aplicar las funciones del algoritmo rijndael para encriptar obtenemos "11101011" (EB), que es expresado en los leds.

El valor mostrado en el LCD es el valor des encriptado, que en este caso el programa va guardando los datos escritos durante el proceso de des encriptación, el ultimo valor mostrado en el LCD es el que fue encriptado en este caso la letra U, para iniciar la des encriptación, se tiene que habilitar por medio del SW0 como indica la *figura 23,* también se puede por el programa en LabVIEW con un Switch para iniciar el proceso de des encriptado como se muestra en la *figura 24.*

Figura No. 24: *Interfaz Grafica en Labview, muestra el resultado de la encriptación y des encriptación*

En la *figura 24,* se observa la interfaz grafica de LabVIEW para el proceso de encriptado y Des Encriptado. La ventana de encriptación nos muestra en la etiqueta "key presionado" el valor hexadecimal que mandamos con el teclado, también cuenta con un contador de dígitos que van siendo usados en el LCD, el resultado de la encriptación es mostrado en la etiqueta de "key encriptado" de forma hexadecimal y binaria con los indicadores azules. La interfaz para el des encriptado nos muestra dos etiquetas, la primera es el valor encriptado y la segunda es el dato de salida.

7.1 DESCRIPCIÓN DEL SISTEMA

El sistema cuenta con programa desarrollado en Visual Basic e instalado en la PC, el cual es la interfaz que nos permite comprimir o encriptar por hardware un archivo de texto plano. La comunicación entre el microcontrolador y la PC, se realiza de dos maneras, una de ellas es mediante la comunicación serial de la PC y la otra es conectando un convertidor de USB a serial, en el puerto USB de la PC, a continuación se muestra un diagrama a bloques del sistema implementado con los dos tipos de conexiones:

El sistema implementado en la PC, presenta una interfaz muy amigable y fácil de instalar. La encriptación de los archivos de texto plano esta limitado a 1000 caracteres, la compresión que utiliza la codificación a 4 bits esta habilitada para 1000 caracteres, pero la compresión y encriptación ADFGVX únicamente tiene la capacidad de encriptar 550 letras.

7.2 IMPLEMENTACIÓN EN EL MICROCONTROLADOR

7.2.1 COMPRESIÓN

7.2.1.1 Compresión utilizando codificación a 4 bits

El sistema de compresión es capaz de comprimir textos que tengan 1000 caracteres, los cuales estén basados en el siguiente alfabeto: **acdeilmnoprstu** el cual pertenece a los caracteres mas utilizados en español según una muestra de texto analizada. La compresión se realiza por bloques de texto que contengan una cantidad de caracteres que sean múltiplos de 10. Para la descompresión, el sistema tiene la capacidad de descomprimir 1000 letras.

Las limitaciones de este tipo de compresión es que el alfabeto es muy corto, no se pueden escribir espacios en blancos.

7.2.2 ENCRIPTACIÓN

7.2.2.1 Algoritmo diseñado por el residente

Este algoritmo tiene la capacidad de encriptar un texto que contenga 1000 caracteres. Los textos deben estar basados en un alfabeto que incluye los siguientes caracteres:

SPC	!	"	#	$	%	_	'	(	)	*	+	,	-	.	/
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

Tabla No. 23: *"Alfabeto para encriptación"*

7.2.2.2 Algoritmo Gronsfeld

Este algoritmo puede encriptar un conjunto de caracteres que se encuentren escritos en mayúsculas, y tiene la particularidad que acepta los espacios en blanco, y no deja patrones en el texto encriptado lo que nos brinda mayor seguridad en el momento de escribir por que, cada espacio en blanco del archivo a encriptar es sustituido por una letra al azar que proporciona el microcontrolador, la cual varia de las operaciones que se hayan realizado en el microcontrolador. Los textos que se deseen encriptar bajo este algoritmo deben estar basados en el siguiente alfabeto:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
		P	Q	R	S	T	U	V	W	X	Y	Z		
		80	81	82	83	84	85	86	87	88	89	90		

Tabla No. 24: *"Alfabeto de algoritmo Gronsfeld"*

Las letras minúsculas que se encuentren en el archivo a encriptar, son cambiadas a mayúsculas automáticamente para evitar conflictos entre la PC y el microcontrolador.

9.2.2.3 Encriptación y compresión ADFGVX

Este tipo de encriptación tiene la ventaja de que utiliza las letras minúsculas y 10 dígitos dando como resultado un alfabeto de 26 caracteres.

			0	1	2	3	4	5	6	7	8	9		
			48	49	50	51	52	53	54	55	56	57		
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
		p	q	r	s	t	u	v	w	x	y	z		
		112	113	114	115	116	117	118	119	120	121	122		

Tabla No. 25: *"Alfabeto para la encriptación y compresión"*

La desventaja que nos ofrece este tipo de encriptación es que nos devuelve el doble del tamaño del archivo de texto a encriptar, es decir, si tenemos un archivo de texto con 10 letras, después de aplicarle este algoritmo nos entregara un texto encriptado el cual tendrá un tamaño de 20 letras, esto puede representar un problema pequeño aparentemente, pero reduce la capacidad de encriptación del sistema. Para resolver este problema pues simplemente le aplicamos la codificación a 4 bits, es decir, le aplicamos nuestro sistema de compresión, y de esta manera podemos comprimir alrededor de 550 letras.

7.3 DESCRIPCIÓN DEL PROGRAMA DEL MICROCONTROLADOR

El programa del microcontrolador fue desarrollado en mikroC 8.2, el microcontrolador se comunica con la PC, a través del puerto. EL puerto serie esta configurado para enviar-recibir 8 bits, sin bit de paridad, con un bit de inicio y una velocidad de transmisión de 19200 bauds. El programa tiene la capacidad de comprimir y encriptar 1000 caracteres, utilizando tres algoritmos y el cuarto algoritmo, que comprime y encripta esta limitado únicamente a 550 caracteres. A continuación se muestra un diagrama del programa instalado en el microcontrolador 18F4550 de la familia 18F, desarrollado por Microchip:

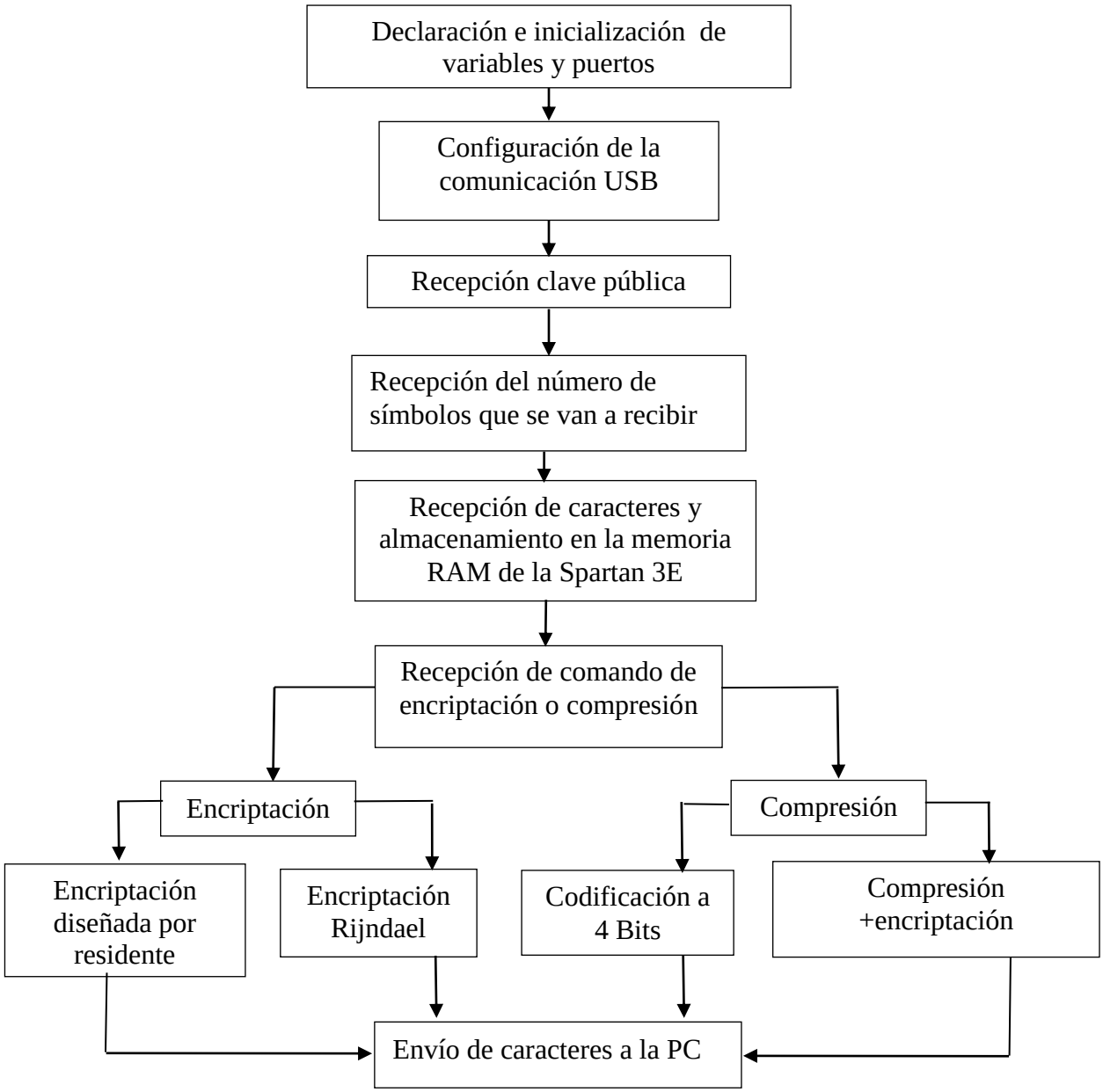


Figura No. 15 *"Diagrama de flujo del programa instalado en el microcontrolador"*

CONCLUSIÓN

El actual trabajo se presento una metodología para adquirir datos desde el puerto PS/2 para la encriptación, des-encriptación y compresión de datos. Para ver la velocidad de respuesta a estos algoritmos, donde se concluye que el FPGA es una arquitectura viable para el uso de estos algoritmos, donde se concluye que el FPGA es una arquitectura viable para el uso de la criptologia y compresión de datos, utilizando una computadora personal solamente para tener una interfaz grafica y comunicación con la tarjeta Spartan 3E en LabVIEW 8.6.

En este proyecto el proceso de adquisición resulto sencillo ya que los datos llegaban en paquetes de dos bytes en forma hexadecimal y solamente había que aplicar las funciones de encriptación y des-encriptación. Con la ayuda de memorias, registros, y multiplexores, ya que son fáciles de programarse en un FPGA, se logran aplicar algunas de las funciones del algoritmo Rijndael.

Las aplicación que se realizaran en un futuro con esta metodología será la implementación del algoritmo rijndael para la encriptación y des-encriptación así como la compresión de los datos.

PROYECCIÓN DEL PROYECTO

La implementación de los algoritmos de encriptación y compresión tiene muchas aplicaciones sobre todo en el campo de la seguridad, hoy toda la información que fluye a través de la red que brinda internet, es susceptible a ser interceptada por agentes externos que no tienen nada que ver con ésta.

Es una manera de blindar información importante y que vaya en menos paquetes de información, al unir ambas herramientas como lo son la compresión y la encriptación. Así vemos reflejado en esta residencia una gran gama de posibles aplicaciones que nos brinda este trabajo.

Por lo que es factible llevar a cabo la continuidad de esta residencia a un proyecto de tesis, con la finalidad de profundizar el estudio en el uso de los FPGA’s y desarrollando diferentes aplicaciones, y seguir con el desarrollo de que refuercen este trabajo en compresión y encriptación de datos.

8. REFERENCIAS BIBLIOGRÁFICAS

[1] Schmelkes Corina, Manual para la presentación de anteproyectos e informes de investigación, 2da Edición, Oxford, 1998.

[2] Pallás, Ramón ; Valdés, Fernando, MICROCONTROLADORES: FUNDAMENTOS Y APLICACIONES CON PIC, 1ª edición (10/02/2007), MARCOMBO EDICIONES TÉCNICAS

[3] ww1.microchip.com/downloads/en/DeviceDoc/ 39632b.pdf

[4] www.steren.com.mx

[5] www.ariellorella.net.des.htm

[6] http://es.wikipedia.org/wiki/Compresión_de_datos

[7] http://www.pbcrypto.com/view.php?algorithm=lz78

[8] http://www.galeon.com/odiseus/index.html

[9] http://www.programacionenc.net

[10] Spartan-3E FPGA Starter Kit Board User Guide

[11] BIHAM E., ¨*A note on Comparing the AES Candidates*¨. Second AES conference, 1999.
http://www.cs.technion.ac.il/~biham/

[12] BLAHUT, R. E., COSTELLO, D. J., MAURER, U., MITTEHOLZER, T., ¨*Communications and Cryptography. Two Sides of One Tapestry*¨. Kluwer Academic Publishers. 1994.

[13] DAEMEN, J., RIJMEN V., ¨*AES Proposal: Rijndael*¨. Document version 2. Date: 03/09/99. NIST´s AES home page. http://www.nist.gov/aes.

[14] DESARROLLO DE ENCRIPTADO AES EN FPGA
Tesis presentada para obtener el grado de
Magíster en Redes de Datos

[15] Seguridad Europea para EEUU.
Algoritmo Criptografico Rijnadael
Autor Alfonso Muñoz Muñoz

ANEXOS