CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL

Unidad Zacatenco

Departamento de Computación

Implementación Eficiente en Hardware Reconfigurable de Esquemas de Cifrado Entonados

Tesis que presenta

Cuauhtemoc Mancillas López

para obtener el Grado de

Maestro en Ciencias

en la Especialidad de

Ingeniería Eléctrica

Opción

Computación

Director de la Tesis

Dr. Debrup Chakraborty

México,DF **22 de Noviembre del 2007**

Resumen

Un esquema de cifrado entonado es un modo de operación de cifradores por bloque que preserva la longitud del texto en claro y que proporciona una permutación pseudoaleatoria fuerte. Se ha sugerido que estos esquemas pueden ser usados como bloque principal para lograr la construcción de cifradores con seguridad demostrable para medios de almacenamiento grandes tales como discos duros. En los últimos años, se ha intensificado la investigación en la construcción de esquemas de cifrado entonados seguros y eficientes. Sin embargo, hasta el día de hoy no se han reportado implementaciones ni los rendimientos correspondientes. Respondiendo a esta ausencia de información se decidió presentar en esta tesis la implementación optimizada de 5 esquemas de cifrado entonados: HCH, HCTR, XCB, EME y TET, los cuales fueron implementados usando el cifrador por bloques AES de 128 bits, utilizado tanto en modo secuencial como en modo tubería (*pipeline*). Asimismo, la función picadillo polinomial universal incluida en la especificación de HCH, HCHfp (una variante de HCH), HCTR, XCB y TET, fue implementada usando un multiplicador de Karatsuba y Ofman como bloque principal. En este documento se proporciona un cuidadoso análisis algorítmico de cada esquema, así como su rendimiento logrado en varios escenarios. Nuestros experimentos muestran que utilizar AES en forma secuencial no es una opción atractiva para el diseño de estos modos ya que propicia tasas de cifrado muy bajas. En contraste, cuando utilizamos AES cifrado/descifrado en modo tubería se logra una tasa de cifrado de 3.67 Gpbs para HCTR y usando AES solo cifrado en modo tubería se logra una tasa de transferencia de 5.71 Gbps para EME. Es así como los resultados del rendimiento proporcionados en esta tesis proporcionan una evidencia experimental de que las implementaciones en hardware de esquemas de cifrado entonados pueden incluso superar las tasas de transferencia de datos logradas por los controladores de disco actuales, lo que demuestra que pueden ser utilizados para construir cifradores internos de discos duros.

Abstract

Tweakable enciphering schemes are length preserving block cipher modes of operation that provide a strong pseudo-random permutation. It has been suggested that these schemes can be used as the main building blocks for achieving in-place disk encryption. In the past few years there has been an intense research activity towards constructing secure and efficient tweakable enciphering schemes. But, actual experimental performance data of these newly proposed schemes are yet to be reported. Accordingly, in this thesis we present optimized FPGA implementations of five tweakable enciphering schemes, namely, HCH, HCTR, XCB, EME and TET, using a 128-bit AES core as the underlying block cipher. We report performance timings of these modes when using both, pipelined and sequential AES structures. The universal polynomial hash function included in the specification of HCH, HCHfp (a variant of HCH), HCTR, XCB and TET, was implemented using a Karatsuba-Ofman multiplier as the main building block. We provide detailed analyses of each of the schemes and their experimental performances achieved in various scenarios. Our experiments show that a sequential AES core is not an attractive option for the design of these modes as it leads to rather poor throughputs. In contrast, by using an encryption/decryption pipelined AES core we get a throughput of 3.67 Gbps for HCTR and by using a encryption only pipeline AES core we get a throughput of 5.71 Gbps for EME. The performance results reported in this thesis provide experimental evidence that hardware implementations of tweakable enciphering schemes can actually match and even outperform the data rates achieved by state-of-the-technology disk controllers, thus showing that they might be used for achieving provably secure in-place hard disk encryption.

Agradecimientos

Agradezco a mis padres por todo el apoyo brindado y por ser ese gran ejemplo para mí, en todos los aspectos de mí vida.

Un agradecimiento especial para mí alma gemela Abril, gracias por la paciencia que me has dado para concluir este trabajo. Tú amor siempre ha sido una motivación para mí. Gracias por todo. Otra etapa que cumplimos juntos, espero que sean más.

A mis compañeros de generación que siempre han estado a mi lado, compartiendo desveladas, preocupaciones y momentos agradables. Un agradecimiento especial a Saúl, a Jorge, a Daniel, Iván y Juan.

Especialmente siempre recordaré con afecto a Victor, Eduardo y Marco.

A todos los profesores del departamento de computación, por sus valiosos conocimientos transmitidos a través de sus clases.

A mí asesor el Dr. Debrup Chakraborty, por su valiosa dirección de este trabajo y por la gran ayuda prestada para su realización, por mostrarme otra enigmática cultura y por ser un gran ejemplo de lo que un científico joven debe ser.

Al Dr. Francisco Rodríguez, por su gran ayuda y valiosas recomendaciones además de facilitar las herramientas necesarias para la realización de esta tesis.

Al Dr. Guillermo Morales, por los comentarios hechos a este trabajo y por ser un gran ejemplo de cultura y sabiduría.

Un agradecimiento especial al Dr. Luis Gerardo de la Fraga y al proyecto CONACyT 45306 por financiar parcialmente este trabajo.

Al CINVESTAV y al CONACyT, por darme la oportunidad de realizar estudios en una institución pública de calidad mundial.

Quiero hacer un reconocimiento a una persona cuya labor es fundamental para el desarrollo de nuestras actividades como estudiantes, el camarada Arcadio. Además de esas

amenas y productivas charlas, donde siempre se daba un espacio para escuchar.

Finalmente quiero agradecer a alguien que nunca va a leer este trabajo, pero que es un ejemplo de honradez y de lucha. Con su ejemplo he visto que aunque el camino sea tremendamente adverso hay que continuar soñando y algún día esos sueños serán realidad (creo que no falta mucho). A mi presidente legítimo Andrés Manuel López Obrador.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Capítulo 1

Introducción

Un cifrador por bloques es una de las primitivas criptográficas más importantes, se basan en principios matemáticos y criptográficos bien conocidos. Debido a su inherente eficiencia , estos se utilizan en muchas aplicaciones que requieren de una alta velocidad de cifrado.

En términos generales un cifrador por bloques se compone de dos algoritmos estrechamente relacionados: El cifrado y el descifrado. El cifrado toma un bloque de longitud fija (conocido como texto en claro) y lo transforma en otro bloque de la misma longitud (conocido como texto cifrado o cifra) bajo la acción de una determinada clave secreta (conocida como llave) que puede o no tener la misma longitud que el texto en claro. Un cifrador por bloques debe ser invertible en el sentido de que al usar el algoritmo de descifrado con la llave adecuada, siempre debe ser posible recuperar el mensaje original. Ya que se utiliza la misma llave tanto para cifrar como para descifrar los cifradores por bloques son clasificados como una primitiva de llave pública o llave simétrica.

Formalmente un cifrador por bloques se considera seguro si se comporta como una permutación pseudoaleatoria fuerte, es decir, que un adversario no pueda distinguir la salida que produce de una permutación aleatoria. Esta definición de seguridad es muy fuerte, se espera que para toda entrada un cifrador por bloques seguro produzca una salida aleatoria. Desafortunadamente esta seguridad no puede ser probada, pero se puede considerar que un cifrador por bloques es seguro si nadie ha sido capaz de encontrar un ataque contra él.

Un cifrador por bloques por sí mismo únicamente puede cifrar un bloque, sin embargo en las aplicaciones reales se requiere cifrar un flujo de información que generalmente es mayor a un solo bloque. Para resolver este problema se utilizan los modos de operación. Existen diferentes tipos de modos de operación que se utilizan en diversas aplicaciones: sólo cifrado [44], cifrado y autenticación [38], cifrado y autenticación con datos asociados [37] y los modos para cifrado de discos [7].

Los modos de operación de solo cifrado se utilizan cuando únicamente se requiere cifrar información y no es necesario realizar autenticación, ejemplos de estos son: ECB (*Electronic Code Book*), CBC (*Cipher Block Chaining*) y CFB (*Cipher Feedback*) [44].

Los modos de operación de cifrado y autenticación (AE por sus siglas en inglés) se utilizan cuando se requiere confidencialidad de la información y autenticación. El desarrollo

de este tipo de modos de operación es en la actualidad un área abierta ya que se utilizan para cifrar paquetes de información, lo que es muy útil en redes de comunicaciones. Existe una subclasificación de estos modos basada en el número de pasadas de cifrado para obtener la cifra del texto en claro. Los que sólo utilizan una pasada de cifrado se denominan de paso sencillo y si utilizan dos pasadas se les llama de paso doble. De paso sencillo existen: OCB (*Offset Code Book*) [38], IAPM (*Integrity Aware Parallel Mode*) [21], IACBC (*Integrity Aware Cipher Block Chaining*)[20], XCBC [12] Y XECB [12]. De paso doble existen: CCM (*Counter with CBC*) [9], CWC (*Carter Wegman Counter Mode*) [24], GCM (*Galois Counter Mode*) [29] y EAX [2].

El modo de operación de cifrado y autenticación con datos asociados se utilizan para cifrar solo una parte de un mensaje pero al mismo tiempo permiten que se pueda autenticar todo el mensaje. Una aplicación para estos es el cifrado de los paquetes que viajan en una red ya que se cifra la información contenida en el paquete excepto el encabezado pero es necesario autenticar todo el paquete en conjunto. La construcción de este modo se realiza haciendo algunas modificaciones a los modos de cifrado y autenticación. Un ejemplo de este tipo de modo de operación es OCB-PMAC (*Offset Codebook Mode-Parallelizable Message Authentication Code*) [37]

Los modos de cifrado de discos son una aplicación de una familia de modos de operación propuesta en [26] llamada Esquemas de Cifrado Entonados (TES por sus siglas en inglés que se toman de *Twekable Enciphering Schemes* [8]), tienen la característica de no expandir el texto cifrado, es decir, conserva la longitud del texto en claro [16]. Otra característica importante que tienen es que proporcionan una permutación pseudoaleatoria fuerte (SPRP por sus siglas en inglés), ya que cada bloque del mensaje cifrado depende de todo el mensaje. Como estos modos producen una SPRP se puede decir que realizan una autenticación parcial, esto se refiere a que si se cambia al menos un bit del texto cifrado, entonces cuando se intente descifrar el resultado será un documento totalmente aleatorio. Los TES siempre entregan una salida a diferencia de los modos de cifrado y autenticación en los cuales, si el mensaje es cambiado no permiten el acceso al mensaje descifrado.

Un TES es considerado seguro si resulta difícil computacionalmente para un adversario distinguir entre el TES y una permutación aleatoria. Un TES tiene una entrada adicional al mensaje y la llave llamada entonador, que es un parámetro público que aumenta la variabilidad en el sentido que si se tiene un mensaje y se cifra con una misma llave siempre dará el mismo mensaje cifrado, de tal forma que al introducir el afinador si se cifra el mensaje en varias ocasiones con un afinador diferente el mensaje cifrado será diferente.

Las primeras definiciones completas de TES para cifrar mensajes de longitud arbitraria se presentaron en [16]. En [16] se menciona como una aplicación posible para estos modos el cifrado de discos a bajo nivel, donde el algoritmo de cifrado/descifrado reside en el controlador de disco el cual accede a los sectores del disco pero no tiene conocimiento de la estructura del disco en alto nivel como directorios, archivos, etc. El controlador de disco cifra el mensaje antes de ser escrito en un sector específico y descifra el mensaje después de leerlo de un sector. Adicionalmente en [16] se sugiere que la dirección del sector puede ser usada como afinador. Por la naturaleza específica de esta aplicación, se requiere de un esquema de cifrado que preserve la longitud y una permutación pseudoaleatoria fuerte

puede proporcionar la mayor seguridad posible.

En los últimos años se han propuesto numerosos TES. Estas propuestas pueden clasificarse en tres categorías: Cifrado-Máscara-Cifrado, Picadillo-Cifrado-Picadillo y Picadillo-Contador-Picadillo. CMC [16], EME [17] y EME* son del tipo Cifrado-Máscara-Cifrado. PEP [7], TET [15] y HEH [41] son del tipo Picadillo-ECB-Picadillo, XCB [30], HCTR [34], HCH [6] y ABL [32] son del tipo Picadillo-Contador-Picadillo.

Alrededor de diez construcciones de TES han sido propuestas, pero hasta la fecha no han sido publicados datos experimentales sobre estos modos no han sido publicados. De las construcciones fácilmente puede concluirse que los esquemas Cifrado-Máscara-Cifrado usan aproximadamente dos llamados al cifrador por cada bloque del mensaje, mientras que los otros dos tipos requieren un llamado al cifrador y dos multiplicaciones en $GF(2^n)$ donde n es el tamaño de bloque en bits, por cada bloque del mensaje. Desde el punto de vista de eficiencia, se argumenta en [6, 8, 31] que el primer tipo es más lento que los otros dos, esto se basa en que un llamado al cifrador por bloques es más costoso que dos multiplicaciones de campo. Este argumento sin embargo puede sostenerse solo si se asume una implementación del modo en software, donde un llamado al cifrador por bloques siempre tiene el mismo costo sin importar las dependencias de datos. Este no es el caso en hardware, donde un llamado al cifrador por bloques puede tener diferente costo en tiempo de acuerdo a la estrategia elegida para el diseño (usar una arquitectura secuencial o una arquitectura en tubería) y la dependencia de datos asociada con el algoritmo del modo de operación particular bajo análisis. El mismo argumento se mantiene para los multiplicadores, se puede tener un multiplicador eficiente totalmente paralelizado el cual puede multiplicar dos elementos del campo en un solo un ciclo de reloj.

Una comparación especulativa del rendimiento de los modos de operación EME*, XCB, HCH y TET en hardware se hizo en [15]. En este comparativo se asume la misma implementación en hardware reportada en [4], donde una multiplicación totalmente paralelizada es implementada en un ciclo de reloj con un costo en área de aproximadamente tres veces el costo asociado con una ronda de AES, y donde el AES es implementado con una tubería de 10 niveles. Sin embargo, éste análisis podría no ser muy preciso, pues como se verá en esta tesis, es posible implementar un multiplicador en $GF(2^n)$ con una eficiencia comparable al de una ronda de AES en términos de su ruta crítica y del costo en área.

Teniendo en cuenta el objetivo de la aplicación específica de cifrado de disco en bajo nivel, un estudio comparativo del rendimiento y el costo en hardware de los esquemas propuestos es muy necesario. La reciente normalización para dichos modos por el grupo sobre seguridad en almacenamiento [18] tambien exige información sobre el rendimiento de muchos de los esquemas propuestos.

En esta tesis se presenta la implementación optimizada en hardware de cinco TES. Los modos escogidos son HCH, HCTR, XCB, EME y TET. También se proporciona el rendimiento de una variante de HCH, llamada HCHfp, la cual es particularmente útil para el cifrado de disco. El fundamento de la elección de estos modos se discute a continuación.

Los modos que se han dejado fuera de este estudio son CMC, PEP, EME*, ABL y HEH. CMC utiliza dos pasos de cifrado con CBC el cual no se puede implementar en tubería. PEP y ABL son particularmente ineficientes comparados con sus homólogos. EME* es

una modificación de EME que maneja mensajes de longitud arbitraria, para la aplicación de cifrado de disco esta funcionalidad no es necesaria. HEH es una propuesta reciente la cual mejora a TET. Se supone que HEH es mejor que TET en ciertos escenarios [41].

1.1. Organización de la Tesis

La tesis esta organizada de la siguiente manera:
En el capítulo 2, se explican ampliamente los TES que se implementaron en la presente tesis. Se muestran los algoritmos y las operaciones matemáticas necesarias para su implementación. Finalmente se hace un comparativo respecto al costo computacional de cada unos de ellos. En el capítulo 3, se describe el algoritmo AES y el multiplicador de Karatsuba y Ofman. En el capítulo 4, se trata lo referente a la implementación del AES y el multiplicador de Karatsuba y Ofman. En el capítulo 5, se describe detalladamente la implementación de los 6 esquemas de cifrado entonados realizadas en esta tesis. En el capítulo 6, se muestran y se analizan los resultados obtenidos. También se hace un comparativo especulativo con soluciones en software existentes en la literatura abierta. En el capítulo 7, se exponen las conclusiones de esta tesis.

Capítulo 2

Esquemas de Cifrado Entonados

En este capítulo se define los Esquemas de Cifrado Entonados, se discuten los requerimientos de seguridad, se presentan las propuestas existentes dentro de esta familia de modos de operación de cifradores por bloque y se explican a detalle los que fueron implementados en la presente tesis.

Las propuestas existentes de *TES* se dividen en tres grupos dependiendo de su construcción. El primero son los esquemas *Picadillo-Contador-Picadillo*, el segundo corresponde a los esquemas *Cifrado-Mezcla-Cifrado* y el tercero son los esquemas *Picadillo-Cifrado-Picadillo*.

2.1. Definiciones

El espacio de mensaje $\mathcal{M}$ es un conjunto de cadenas $\mathcal{M} = \cup_{i \in I}\{0,1\}^i$ para el conjunto de índices no vacío $I \subseteq \mathbb{N}$. $\mathcal{K}$ y $\mathcal{T}$ son conjuntos finitos no vacíos llamados espacio de llaves y espacio de afinador respectivamente. Un cifrador por bloques es una primitiva criptográfica definida como $\mathbf{E} : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ donde n es el tamaño de bloque, se denota como E_K y su inversa E_K^{-1} donde $K \in \mathcal{K}$. Un *TES* es una función $\mathbf{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$, para toda $K \in \mathcal{K}$ y $T \in \mathcal{T}$ y se denota como E_K^T y su inversa D_K^T. $P = P_1 ... P_m$ es un mensaje con m bloques de tamaño n. La función $pad_{n-r}(P_m)$ indica que si el bloque P_m contiene menos de n bits se completa con r $0s$, $drop_{n-r}(P_m)$ indica que se tome la longitud original sin los r $0s$ agregados, $bin(l)$ es la representación binaria de l, $|A|$ es la longitud en bits de A y $A||B$ es la concatenación de las cadenas A y B. Una función picadillo es definida como: $\{0,1\}^{mn} \to \{0,1\}^n$ donde mn es el tamaño del mensaje.

Un campo $GF(2^n)$ contiene todas las cadenas de $\{0,1\}$ cuya longitud sea igual a n. Sean A y B elementos del campo, la suma y la resta están definidas como $A \oplus B$, $A + B$ denota una suma entera. La multiplicación se realiza utilizando la definición de la suma y está definida como $AB \ mod \ p(x)$ donde $p(x)$ es un polinomio irreducible en el campo.

2.2. Requerimientos de Seguridad para los Esquemas de Cifrado Entonados

Como se mencionó en el capítulo 1 un modo de operación se utiliza para cifrar mensajes cuya longitud es mayor que el tamaño de palabra manejada por el cifrador por bloque. Ahora se analizarán algunos modos de operación para ver si son seguros. Se tratará de formular de forma intuitiva los requerimientos de seguridad para los esquemas de cifrado de disco.

Se comenzará analizando el modo ECB. El modo ECB no es recomendable para cifrar un flujo de información, ya que el texto cifrado revela mucha información sobre el texto en claro. En la figura 2.1 (a) se muestra una imagen y en la figura 2.1 (b) se muestra la imagen cifrada utilizando ECB. Es posible observar que la imagen cifrada revela mucha información acerca de la imagen original, la razón de esto es que se utiliza la misma llave y se cifran igual todos los bloques. Lo anterior provoca que si se cifra un mensaje de cuatro bloques P_1, P_2, P_3, P_4 donde $P_1 = P_2$, entonces en el mensaje cifrado C_1, C_2, C_3, C_4 también C_1 será igual a C_2. Lo anterior no es deseable, en el mensaje cifrado se da mucha información acerca del mensaje original, en el ejemplo anterior el adversario puede saber que los primeros dos bloques del mensaje son iguales sólo con observar el texto cifrado. Esta limitación del cifrado con ECB se muestra en la figura 2.1 (b), la imagen cifrada es similar a la imagen original[1]. Por lo tanto, ECB es inseguro.

Una pregunta importante que se abordará ahora es, cuándo un modo puede ser considerado seguro. Para responder lo anterior primero es necesario establecer un modelo formal del adversario que intenta romper un modo de operación.

2.2.1. El Adversario

Para definir la seguridad se necesita describir los objetivos y recursos de un adversario. Un adversario puede tener varios objetivos, el más fuerte, consiste en recuperar la llave del esquema de cifrado utilizado. Con el conocimiento de llave puede descifrar todos los mensajes cifrados que circulan por un canal público y también puede reemplazar mensajes cifrados con mensajes de su elección. Pero el problema de la recuperación de la llave es un objetivo muy difícil y sin la recuperación de la llave un adversario también puede descifrar o cifrar ciertos mensajes. El objetivo más débil que un adversario puede tener es el de distinguir entre el texto cifrado y cadenas aleatorias. Aunque no es claro lo que un adversario puede hacer con distinguir texto cifrado de cadenas aleatorias, pero si se muestra que un algoritmo de cifrado es seguro ante estos ataques básicos, entonces estamos ciertos de que el adversario no puede montar ataques con objetivos más fuertes.

Un adversario que quiere romper la seguridad de un sistema criptográfico simétrico debe tener acceso a algunas de las entradas o salidas del sistema. El tipo de información a la que accede define su poder. Él siempre tiene acceso al texto cifrado ya que puede escuchar el canal público. Si sólo tiene acceso al texto cifrado, el ataque montado por él

[1]Ejemplo tomado de <http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation>.

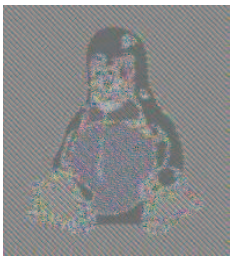
(a)  (b) 

Figura 2.1: (a) Imagen Original. (b) Imagen cifrada con ECB.

se llama de sólo texto cifrado. Si además puede conocer los mensajes que producen estos textos cifrados, es un ataque de texto en claro conocido. Si puede elegir el texto en claro de los textos cifrados que quiera entonces es un ataque de texto en claro elegido. Puede haber adversarios que puedan elegir el texto cifrado y obtener el correspondiente texto en claro para éstos, a este se le llama ataque de texto cifrado elegido. El adversario más fuerte es el que puede elegir mensajes de forma adaptable (texto cifrado) y obtener sus textos en claro (descifrado). Esos son llamados adversarios que eligen el texto en claro (respectivamente el texto cifrado).

Con el propósito de formalizar se verá al adversario como un algoritmo probabilístico en tiempo polinomial, con determinados recursos. Una elección adaptable del texto en claro del adversario debe ser proporcionada con textos cifrados de su elección. Para ello se permite al adversario comunicarse con el algoritmo de cifrado, es decir, éste tiene acceso al esquema de cifrado como una caja negra donde proporciona algunas entradas y obtiene la correspondiente salida pero no tiene acceso al funcionamiento interno del esquema. Este tipo de solicitudes es conocida como el acceso a un oráculo. Un adversario puede tener acceso a uno o más oráculos, como en el caso de un adversario con elección adaptable del texto en claro y elección del texto cifrado, se le debe dar al adversario acceso a los oráculos de cifrado y de descifrado para que pueda obtener los cifrados y descifrados de su elección.

2.2.2. Seguridad de los Esquemas para Cifrado de Discos

Supongamos que se quiere cifrar toda la información presente en el disco duro de una computadora. Siempre que hay una lectura del disco, un sector particular del disco es descifrado y entregado, similarmente siempre que algunos datos necesitan ser escritos en el disco se cifran y se escriben en un sector específico del disco. Las operaciones de cifrado y descifrado son realizadas por el controlador de disco, el cual es un dispositivo de bajo nivel que no tiene conocimiento de archivos, directorios, etc. que son estructuras manejadas por el sistema operativo.

En este entorno una limitación muy importante es que el algoritmo usado para cifrar debe preservar la longitud, es decir, la longitud del texto cifrado debe ser igual a la longitud del texto en claro. Esta limitación no permite que los esquemas de cifrado y autenticación puedan ser utilizados en esta aplicación, ya que éstos siempre producen una expansión en el mensaje cifrado. Los modos de sólo cifrado en general preservan la longitud, pero la seguridad que proporcionan no es suficiente para esquemas de cifrado de discos, pues

no se quiere a un adversario manipulando los datos presentes en el disco sin nuestro consentimiento. Entonces, se necesitan esquemas en los cuales el texto cifrado producido sea indistinguible de cadenas aleatorias para cualquier adversario que pueda accesar a la cifra correspondiente a los mensajes de su elección. Además si un adversario elige cifras y obtiene los textos en claro correspondientes debe ser incapaz de distinguir estos de cadenas aleatorias.

En estos esquemas se da la libertad al adversario de elegir también textos cifrados. Este adversario es un adversario con elección adaptable del texto en claro y con elección del texto cifrado. Así, en este escenario, si el adversario planea cambiar el texto cifrado original en el disco con algunos textos cifrados de su elección, entonces el texto en claro que obtenga será indistinguible de cadenas aleatorias. De esta manera, el adversario no podrá crear ningún texto cifrado el cual le proporcione información importante, en otras palabras con todos los textos cifrados creados por el adversario cuando los descifre sólo verá cadenas aleatorias, y es posible que una aplicación de alto nivel detecte que el adversario ha cambiado el texto cifrado. Los esquemas de cifrado entonados cumplen estos requerimientos de seguridad, es decir, son permutaciones pseudoaleatorias fuertes SPRPs (por sus siglas en ingles).

En la siguiente sección se describen los TES que fueron implementados en esta tesis, de acuerdo al tipo al que pertenecen de acuerdo a su estructura.

2.3. Picadillo-Contador-Picadillo

Esta construcción fue introducida por McGrew y Fluhrer en [30] donde presentan el modo de operación XCB. La estructura general de estos esquemas, se compone de un cifrador por bloques en modo contador colocado entre dos funciones picadillo universales. Las funciones picadillo son del tipo Wegman-Carter, que son funciones polinomiales [5]. A continuación se detallarán tres propuestas de este tipo de construcciones: XCB, HCTR [34], HCH [6].

2.3.1. XCB

Este modo de operación fue propuesto por McGrew y Fluhrer en 2004, llamado XCB por sus siglas del inglés que se toman de *Extended Code Book* [30]. En la figura 2.2 se muestra su diagrama a bloques.

El esquema sólo utiliza una llave K, pero genera 5 llaves internamente de la siguiente forma: $K_0 = E_K(0^n)$, $K_1 = E_K(0^{n-1}||1)$,$K_2 = E_K(0^{n-2}||10)$, $K_3 = E_K(0^{n-2}||11)$ y $K_4 = E_K(0^{n-3}||100)$.

El modo contador para este esquema se define como: $Ctr_{K,S}(P) = P_1 \oplus E_K(S)||P_2 \oplus E_K(incr(S))||...||P_m \oplus MSB_t(E_K(incr^{w-1}(S)))$ donde K es la llave, S es un vector de inicialización, $MSB_t(S)$ es una función que regresa los t bits más significativos de S, $w = \lceil m/n \rceil$ es el número de bloques completos, $t = m \ mod \ n$ es el número de bits en el último bloque y la función $incr(S)$ se define como sigue,

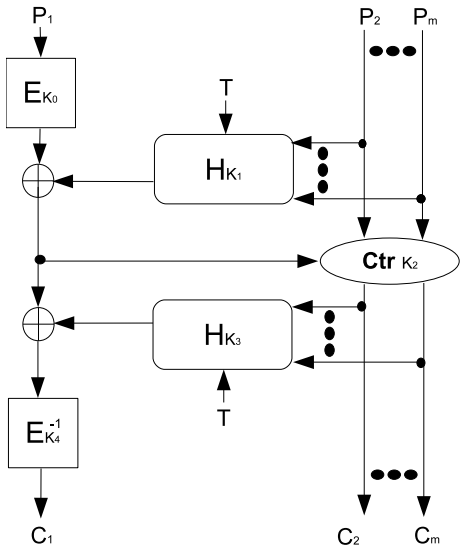


Figura 2.2: Diagrama de bloques de XCB.

$$incr(S) = S[0, w-33] || \left([w-32, w-1] + 1 \ mod \ 2^{32}\right) \tag{2.1}$$

donde la cadena S está numerada de izquierda a derecha. La función $incr(S)$ genera incrementos sucesivos de S módulo 2^{32}, de esta manera el XCB esta limitado a mensajes por el modo contador a manejar mensajes 2^{39} bits. En el algoritmo 1 se muestra la forma de implementarlo.

Algoritmo 1 Modo contador de XCB

Entrada: K,S y $P_1, ...P_m$.
Salida: Mensaje cifrado $Ctr(P)$.

1: $w = \lceil m/n \rceil$
2: **para** $i \leftarrow 1$ **hasta** $w-1$ **hacer**
3: $\quad C_i \leftarrow P_i \oplus E_k(incr^{i-1}(S))$
4: $t = m \ mod \ n$
5: $C_m = MSB_t(P_w \oplus E_k(incr^{w-1}(S)))$
6: **regresar** $C_1 ... C_m$

La función picadillo de XCB $h(K, P, T)$ es la que se utiliza en el GCM (*Galois Counter Mode*) [29] y se define en (2.2).

$$X_i = \begin{cases} 0 & \text{si } i = 0 \\ (X_{i-1} \oplus P_i) \cdot K & \text{si } i = 1...m-1 \\ (X_{m-1} \oplus pad_{n-r}(P_m)) \cdot K & \text{si } i = m \\ (X_{i-1} \oplus T_{m-1}) \cdot K & \text{si } i = m+1...m+q-1 \\ (X_{m+q-1} \oplus pad_{n-r}(T_q)) \cdot K & \text{si } i = m+q \\ (X_{m+q} \oplus (bin(|P|)||bin(|T|))) \cdot K & \text{si } i = m+q+1 \end{cases} \tag{2.2}$$

El algoritmo 2 corresponde a la función $h(K,P,T)$, para la evaluación de polinomio se utiliza la regla de Horner.

Algoritmo 2 Función picadillo de XCB

Entrada: $P_1,...,P_m$,K y T.
Salida: $C_1,...,C_m$.

$X_0 \leftarrow 0^n$
para $i \leftarrow 1$ **hasta** $m-1$ **hacer**
    $X_i \leftarrow (X_{i-1} \oplus P_i) \cdot K$
$X_m \leftarrow (X_{m-1} \oplus pad_{n-r}(P_m)) \cdot K$
para $i \leftarrow m+1$ **hasta** $q-1$ **hacer**
    $X_i \leftarrow (X_{i-1} \oplus T_{i-m}) \cdot K$
$X_{n+q} \leftarrow (X_q \oplus pad_{n-r}(T_q)) \cdot K$
$X_{m+q+1} \leftarrow (X_{m+q} \oplus pad_{n-r}(bin(|A|)||bin(|T|))) \cdot K$
regresar X_{m+q+1}

Usando la definición para el modo contador y para la función picadillo en el algoritmo 3 se muestra el cifrado y en el algoritmo 4 el descifrado utilizando XCB.

Algoritmo 3 Cifrado XCB

Entrada: $P_1,...,P_2$,K y T.
Salida: $XCB_{K,T}(P)$.

$K_0 \leftarrow E_K(0^n)$
$K_1 \leftarrow E_K(0^{n-1}||1)$
$K_2 \leftarrow E_K(0^{n-2}||10)$
$K_3 \leftarrow E_K(0^{n-2}||11)$
$K_4 \leftarrow E_K(0^{n-3}||100)$
$D \leftarrow E_{K_0}(P_1)$
$H1 \leftarrow D \oplus h(K_1,P_2...P_m,T)$
$C_2...C_m \leftarrow Ctr(K_2,H1,P_2...P_m)$
$H_2 \leftarrow H1 \oplus h(K_3,C_2...C_m,T)$
$C_1 \leftarrow E^{-1}_{K_4}(H_2)$
regresar $C_1,...,C_m$

Algoritmo 4 Descifrado XCB

Entrada: $C_1,...,C_m$,K y T.
Salida: $P_1,...,P_m$.

1: $K_0 \leftarrow E_K(0^n)$
2: $K_1 \leftarrow E_K(0^{n-1}||1)$
3: $K_2 \leftarrow E_K(0^{n-2}||10)$
4: $K_3 \leftarrow E_K(0^{n-2}||11)$
5: $K_4 \leftarrow E_K(0^{n-3}||100)$
6: $D \leftarrow E_{K_4}(C_1)$
7: $H1 \leftarrow D \oplus h(K_3, C_2...C_m, T)$
8: $P_2...P_m \leftarrow Ctr(K_2, H2, C_2...C_m)$
9: $H_2 \leftarrow H1 \oplus h(K_1, P_2...P_m, T)$
10: $P_1 \leftarrow E_{K_0}^{-1}(H_2)$
11: **regresar** $P_1,...,P_m$

Respecto al costo computacional de XCB: Si se considera un mensaje de m bloques cada uno de tamaño n y que el afinador T es sólo un bloque de tamaño n, el costo es de $2(m+1)$ multiplicaciones en $GF(2^n)$ y $m+6$ llamadas al cifrador por bloques. El esquema requiere sólo una llave de usuario. El costo computacional para el cifrado y el descifrado es el mismo.

2.3.2. HCTR

Este modo de operación fue propuesto por Wang et al. en el 2004. HCTR por las siglas tomadas de su nombre en inglés *Hash-Counter* [34]. Utiliza una llave única, dos funciones picadillo universales, un cifrador por bloques en modo contador y un cifrador por bloques en modo simple, es decir, para cifrar un solo bloque. Este modo de operación utiliza dos llaves K para AES y h para las funciones picadillo. Su estructura se muestra en la figura 2.3

La función picadillo se define en 2.3 y se calcula como se muestra en el algoritmo 5.

Algoritmo 5 Función picadillo de HCTR

Entrada: $P_1,...,P_m$ y h.
Salida: $H(h,P)$.

1: $X_0 \leftarrow 0^n$
2: **para** $i \leftarrow 1$ **hasta** $m-1$ **hacer**
3: $\quad X_i \leftarrow (X_{i-1} \oplus P_i) \cdot h$
4: $X_m \leftarrow (X_{m-1} \oplus pad_{n-r}(P_m)) \cdot h$
5: $X_{m+1} \leftarrow (X_m \oplus bin(|P|)) \cdot h$
6: **regresar** X_{m+1}

$$H(h,P) \leftarrow P_1 \cdot h^{m+1} \oplus ... \oplus pad_{n-r}(P_m) \cdot h^2 \oplus len(P) \cdot h \tag{2.3}$$

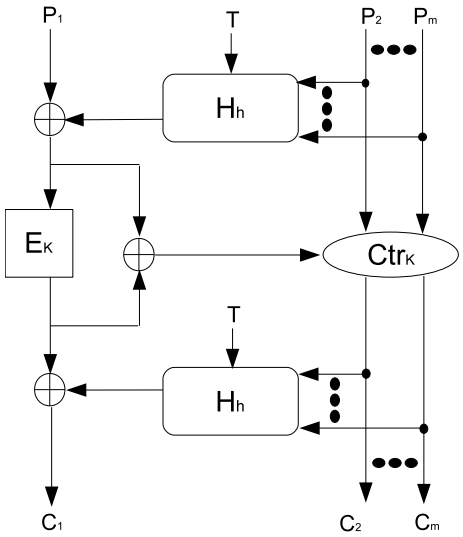


Figura 2.3: Diagrama de bloques de HCTR.

El cifrador en modo contador para HCTR se define como: $Ctr_{K,S}(P) = P_1 \oplus E_K(S \oplus bin(1))||...||P_m \oplus E_K(S \oplus bin(m))$ donde K es la llave, S es un vector de inicialización y se calcula como se muestra en el algoritmo 6. Se puede observar que el afinador T se concatena con la entrada de la función picadillo H.

Algoritmo 6 Modo contador de HCTR

Entrada: K,S y P.
Salida: $C_1,...,C_m$.

1: **para** $i \leftarrow 1$ **hasta** m **hacer**
2: $\quad C_i \leftarrow P_i \oplus E_k(S \oplus i)$
3: $drop_{n-r}(C_m)$
4: **regresar** $C_1...C_m$

En el algoritmo 7 se muestra la operación de cifrado de HCTR y en el algoritmo 8 la operación de descifrado.

Respecto al costo computacional de HCTR: Si se considera un mensaje de m bloques cada uno de tamaño n y que el afinador T es sólo un bloque de tamaño n, el costo computacional es de $2(m+1)$ multiplicaciones en $GF(2^n)$ y m llamados al cifrador por bloques. El esquema requiere dos llaves de usuario. El costo computacional para el cifrado y el descifrado es el mismo.

Algoritmo 7 Cifrado HCTR

Entrada: $P_1,...,P_m$, k,h y T.
Salida: $C_1,...,C_m$.
1: $MM \leftarrow P_1 \oplus H_h(P_2...P_m||T)$
2: $CC \leftarrow E_k(MM)$
3: $S \leftarrow MM \oplus CC)$
4: $C_2...C_m \leftarrow CTR_k^S(P_2...P_m)$
5: $C_1 \leftarrow CC \oplus H_h(C_2...C_m||T)$
6: **regresar** $C_1,...,C_m$

Algoritmo 8 Descifrado HCTR

Entrada: $C_1,...,C_m$,T,K y h.
Salida: $P_1,...,P_m$.
1: $CC \leftarrow C_1 \oplus H_h(C_2...C_3||T)$
2: $MM \leftarrow E_k^{-1}(CC)$
3: $S \leftarrow MM \oplus CC$
4: $P_2...P_m \leftarrow CTR_k^S(C_2...C_m)$
5: $P_1 \leftarrow MM \oplus H_h(P_2...P_m||T)$
6: **regresar** $P_1,...,P_m$

2.3.3. HCH

Este modo de operación fue propuesto por Chakraborty y Sarkar en el 2006, HCH por la siglas tomadas de su nombre en inglés que provienen de *Hash-Counter-Hash* [6]. A diferencia de HCTR éste utiliza una única llave K, de esta utilizando el afinador T genera la llave $R = E_K(T)$ y calcula otro parámetro $Q = E_K(R \oplus len(P))$ para utilizarse en las funciones picadillo. El esquema se muestra en la figura 2.4.

El cifrador por bloques en modo contador es el mismo que utiliza HCTR y que se describió en el algoritmo 5. La función picadillo se define en la ecuación (2.4) y se calcula como se muestra en el algoritmo 9.

$$H(P,R,Q) \leftarrow P_1 \oplus Q \oplus P_2 \cdot R^{m-1} \oplus ... \oplus P_m \cdot R \tag{2.4}$$

Algoritmo 9 Función picadillo de HCH

Entrada: P y h.
Salida: $H(R,Q,P)$.
1: $X_0 \leftarrow 0^n$
2: **para** $i \leftarrow 2$ **hasta** m **hacer**
3: $\quad X_{i-1} \leftarrow (X_{i-2} \oplus P_i) \cdot R$
4: $X_m \leftarrow X_{m-1} \oplus P_1$
5: $X_{m+1} \leftarrow X_m \oplus Q$
6: **regresar** X_{m+1}

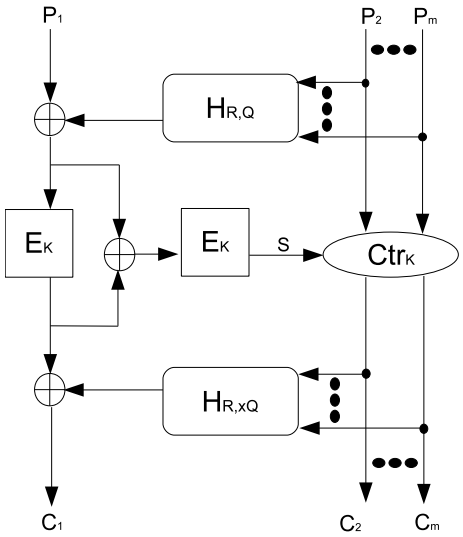


Figura 2.4: Diagrama de bloques de HCH.

El cifrado utilizando HCH se muestra en el algoritmo 10 y el descifrado en 11.

Algoritmo 10 Cifrado HCH

Entrada: $P1,...,P_m$, K y T.
Salida: $C_1,...,C_m$.

$R \leftarrow E_K(T)$; $Q \leftarrow E_K(R \oplus bin(|P|))$
$M_1 \leftarrow H_{R,Q}(P_1,...,P_m)$
$U_1 \leftarrow E_K(M_1)$; $I \leftarrow M_1 \oplus U_1$
$S \leftarrow E_K(I)$
$(C_2,...,C_m) \leftarrow Ctr_{K,S}(P_2,...,P_m)$
$C_1 \leftarrow H_{R,xQ}(U_1,C_2,...,C_m)$
regresar$(C_1,...,C_m)$

Algoritmo 11 Descifrado HCH

Entrada: $C_1...C_m$, K y T.
Salida: Mensaje descifrado $P_1,...,P_m$.

1: $R \leftarrow E_K(T)$; $Q \leftarrow E_K(R \oplus bin(|P|))$
2: $U_1 \leftarrow H_{R,Q}(C_1,...,C_m)$
3: $M_1 \leftarrow E_K^{-1}(U_1)$; $I \leftarrow M_1 \oplus U_1$
4: $S \leftarrow E_K(I)$
5: $(P_2,...,P_m) \leftarrow Ctr_{K,S}(C_2,...,C_m)$
6: $P_1 \leftarrow H_{R,xQ}(M_1,P_2,...,P_m)$
7: **regresar**$(P_1,...,P_m)$

El modo HCH originalmente fue propuesto para trabajar con mensajes de longitud variable, el problema de cifrado de discos se reduce a cifrar un mensaje con longitud fija. Los autores de HCH propusieron en el 2007 una mejora de su esquema que se diseñó específicamente para el problema de cifrado de discos. Por tratarse de un mensaje de longitud fija no es necesario que se tome la longitud del mismo dentro del esquema, lo que reduce un llamado al cifrador por bloque pero incrementa a dos el número de llaves requeridas. Este esquema recibe el nombre de HCHfp [8], el algoritmo 12 corresponde al cifrado bajo este esquema.

Algoritmo 12 Cifrado HCHfp

Entrada: $P1,...,P_m$, α K y T.
Salida: $C_1,...,C_m$.
1: $R \leftarrow E_K(T)$
2: $M_1 \leftarrow H_{\alpha,R}(P_1,...,P_m)$
3: $U_1 \leftarrow E_K(M_1)$; $I \leftarrow M_1 \oplus U_1$
4: $S \leftarrow E_K(I)$
5: $(C_2,...,C_m) \leftarrow Ctr_{K,S}(P_2,...,P_m)$
6: $C_1 \leftarrow H_{\alpha,xR}(U_1,C_2,...,C_m)$
7: **regresar**$(C_1,...,C_m)$

Respecto al costo computacional de HCH y HCHfp: Si se considera un mensaje de m bloques cada uno de tamaño n y que el afinador T es sólo un bloque de tamaño n, el costo computacional es de $2(m-1)$ multiplicaciones en $GF(2^n)$ y $m+3$ llamados al cifrador por bloques. Para HCHfp se mantiene el número de multiplicaciones $2(m-1)$, pero el número de llamados al cifrador por bloques disminuye en uno quedando en $m+2$. Para HCH se requiere una llave de usuario y para HCHfp se requieren dos. El costo computacional para el cifrado y el descifrado es el mismo en ambos esquemas.

2.4. Cifrado-Mezcla-Cifrado

Estos esquemas fueron introducidos por Halevi y Rogaway en [16], donde hacen la propuesta de CMC (*CBC-Mask-CBC*). El cuál se basa en la utilización de un cifrador por bloques en modo CBC (*Cipher-Block-Chaining*) y el uso de operaciones simples como sumas en $GF(2^m)$. CBC tiene la desventaja de que su estructura no permite utilizar una arquitectura en tubería ya que existe dependencia entre los los bloques cifrados como se muestra en la figura 2.5.

Posteriormente en [17] los mismos autores proponen EME *ECB-Mix-ECB*, presentado como un modo de operación totalmente paralelizable, este modo se explicará de forma detallada más adelante. Si n es el tamaño de palabra del cifrador por bloques, tanto CMC como EME sólo pueden procesar mensajes cuya longitud en bloques es menor de n bloques. Para manejar mensajes de longitud arbitraria en [14] Halevi presenta una mejora a EME llamada EME$*$.

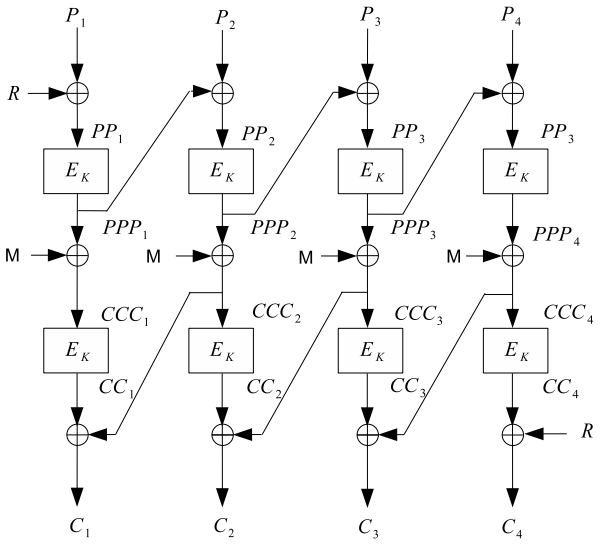


Figura 2.5: Diagrama de bloques de CMC para cuatro bloques, E_K, $M = 2(PPP_1 \oplus PPP_m)$ y $R = E_{\tilde{K}}$.

2.4.1. EME

Este esquema se compone de dos pasos de cifrado con ECB y entre éstas una mezcla compuesta de un llamado al cifrador por bloques, operaciones $\oplus$ y el cálculo de x^iM para $i = 0...m-1$, que hacen que cada una de los bloques cifrados dependa de todo el mensaje en conjunto. Tanto a la entrada como a la salida de EME se realiza una $\oplus$ de cada bloque con x^iL para $i = 0...m-1$. En la figura 2.6 se ilustra EME para cifrar cuatro bloques.

En el algoritmo 13 se muestra el cifrado bajo EME y en el algoritmo 14 el descifrado.

Respecto al costo computacional de EME: Si se considera un mensaje de m bloques cada uno de tamaño n y que el afinador T es sólo un bloque de tamaño n, el costo computacional es de $2m+2$ llamados al cifrador por bloques. El esquema requiere sólo una llave de usuario. El costo computacional para el cifrado y el descifrado es el mismo.

2.5. Picadillo-Cifrado-Picadillo

Este tipo de construcción de *TES* fue propuesta por Naor y Reignold en [33], donde proponen un modo de operación pseudoaleatorio. En éste utilizan dos funciones picadillo y entre éstas un paso de cifrado. En la figura 2.7 se muestra la estructura propuesta por Naor y Reignold.

Las funciones picadillo utilizadas en este tipo de *TES* son de un tipo especial llamadas funciones picadillo universales orientadas a bloque que se definen como $\mathcal{F} : \mathcal{K} \times \mathbb{F}^m \rightarrow \mathbb{F}^m$ y son invertibles para cada $k \in \mathcal{K}$.

Las propuestas existentes actualmente de este tipo de construcciones son PEP *Polihash Encrypt Polihash* [7], TET *Linear-Transformation ECB Linear-Transformation* [15]

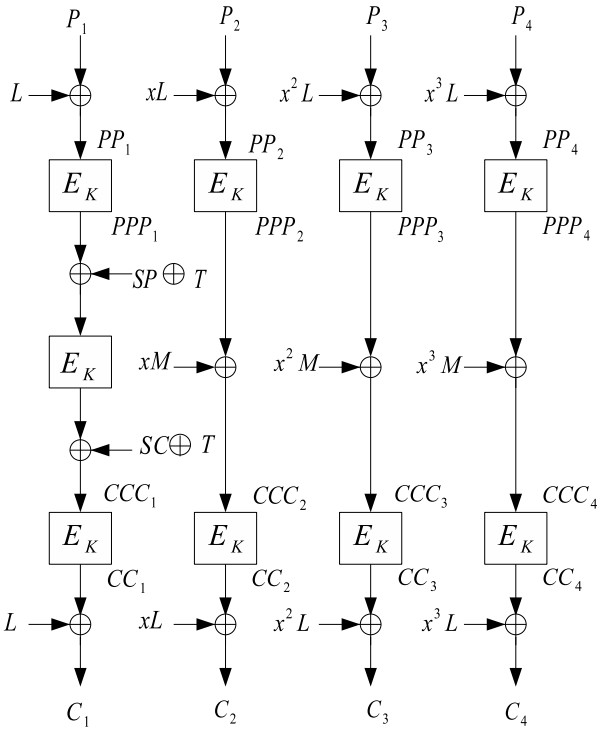


Figura 2.6: Diagrama de bloques de EME para cuatro bloques, $L = 2E_K(0^n)$, $SP = PPP_2 \oplus PPP_3 \oplus PPP_4$, $M = (MP \oplus MC)$ y $SC = CCC_1 \oplus CCC_2 \oplus CCC_3 \oplus CCC_4$.

Algoritmo 13 Cifrado EME

Entrada: $P_1, ..., P_m$, K y T.
Salida: $C_1, ..., C_m$.

1: $L \leftarrow 2E_K(0^n)$
2: **para** $i \leftarrow 1$ **hasta** m **hacer**
3: $\quad PP_i \leftarrow 2^{i-1}L \oplus P_i$
4: $\quad PPP_i \leftarrow E_K(PP_i)$
5: $SP \leftarrow PPP_2 \oplus .... \oplus PPP_m$
6: $MP \leftarrow PPP_1 \oplus SP \oplus T$
7: $MC \leftarrow E_K(MP)$
8: $M \leftarrow MP \oplus MC$
9: **para** $i \leftarrow 2$ **hasta** n **hacer**
10: $\quad CCC_i \leftarrow PPP_i \oplus 2^{i-1}M$
11: $SC \leftarrow CCC_2 \oplus .... \oplus CCC_m$
12: $CCC_1 \leftarrow MC \oplus SC \oplus T$
13: **para** $i \leftarrow 1$ **hasta** m **hacer**
14: $\quad CC_i \leftarrow E_K(CCC_i)$
15: $\quad C_i \leftarrow CC_i \oplus 2^{i-1}L$
16: **regresar**$(C_1, ..., C_m)$

Algoritmo 14 Descifrado EME

Entrada: $P_1, ..., P_m$, K y T.
Salida: $C_1, ..., C_m$.

$L \leftarrow 2E_K(0^n)$
para $i \leftarrow 1$ **hasta** m **hacer**
  $CC_i \leftarrow 2^{i-1}L \oplus C_i$
  $CCC_i \leftarrow E_K(CC_i)$
$SC \leftarrow CCC_2 \oplus .... \oplus CCC_m$
$MC \leftarrow CCC_1 \oplus SC \oplus T$
$MP \leftarrow E_K^{-1}(MC)$
$M \leftarrow MC \oplus MP$
para $i \leftarrow 2$ **hasta** n **hacer**
  $PPP_i \leftarrow CCC_i \oplus 2^{i-1}M$
$SP \leftarrow PPP_2 \oplus .... \oplus PPP_m$
$PPP_1 \leftarrow MP \oplus SP \oplus T$
para i $\leftarrow 1$ **hasta** m **hacer**
  $PP_i \leftarrow E_K^{-1}(PPP_i)$
  $P_i \leftarrow PP_i \oplus 2^{i-1}L$
regresar$(P_1, ..., P_m)$

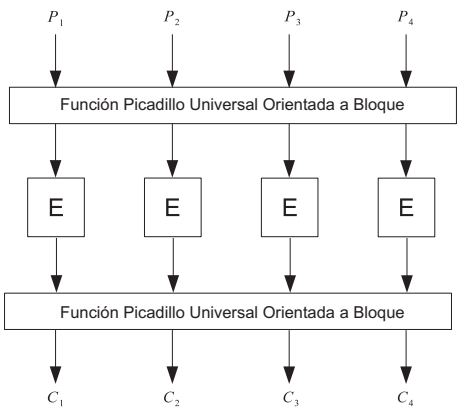


Figura 2.7: Diagrama de bloques de la construcción propuesta por Naor y Reignold.

y posteriormente una mejora propuesta por Sarkar en HEH *Hash Encrypt Hash*[41]. En la presente tesis se implementó el modo de operación TET que se explicara ampliamente a continuación.

2.5.1. TET

Presentado en el 2007 por Halevi en [15], utiliza dos funciones picadillo invertibles orientadas a bloques y entre estas un paso de cifrado con ECB. En la figura 2.8 se muestra el diagrama a bloques de TET.

La función picadillo de TET debe cumplir las siguientes características para poder ser

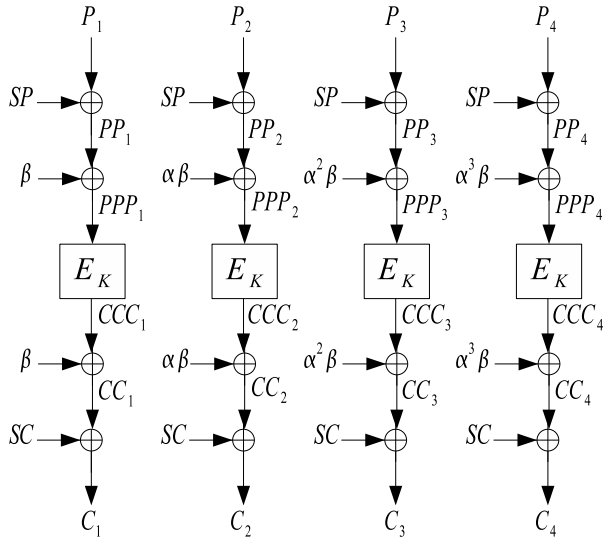


Figura 2.8: Diagrama de bloques de TET para cuatro bloques, *SP* y *SC* se calculan mediante una evaluación polinomial, α es un elemento primitivo en el campo.

invertible: sea $\mathcal{F}$ un campo con más de $m+2$ elementos y considerando una matriz de $m \times m$ elementos sobre $\mathcal{F}$, $M_\tau = A_\tau + I$ para un elemento $\tau \in \mathcal{F}$, donde

$$A_\tau = \begin{bmatrix} \tau & \tau^2 & & \tau^m \\ \tau & \tau^2 & & \tau^m \\ & & \ddots & \\ \tau & \tau^2 & & \tau^m \end{bmatrix} \tag{2.5}$$

El determinante de la matriz A_τ es $\sigma = \sum_{i=0}^{m} \tau^i$, la matriz A_τ es invertible si sólo si $\sigma \neq 0$ y $A_\tau^{-1} = I - (A_\tau/\sigma)$. Es posible calcular $y = M_\tau x$ y $x = M_\tau^{-1} y$ de forma eficiente utilizando una evaluación polinomial. Para $y = M_\tau x$ primero se calcula $s = \sum_{i=1}^{m} x_i \tau^i$ y posteriormente $y_i = s \oplus x_i$, y para su inverso $x = M_\tau^{-1} y$ calculando $s = \sum_{i=1}^{m} y_i (\tau^i/\sigma)$ ó para operar únicamente con multiplicaciones $s = \sigma^{-1} \sum_{i=1}^{m} y_i \tau^i$ y posteriormente $x_i = s \oplus y_i$. En general la función picadillo para TET BPE (*Blockwise Polinomial Evaluation*) se describe como sigue:

$$BPE_{\tau,\beta}(x) = M_\tau x \oplus b \quad y \quad BPE_{\tau,\beta}^{-1}(x) = M_\tau (x \oplus b) \tag{2.6}$$

$$\widetilde{BPE}_{\tau,\beta}(x) = M_\tau (x \oplus b) \quad y \quad \widetilde{BPE}_{\tau,\beta}^{-1}(x) = M_\tau x \oplus b \tag{2.7}$$

Donde b es un vector de m elementos generado como $b_i = \alpha^{i-1}\beta$ para $i = 1 \ldots m$ y $\beta \in \mathcal{F}$ es un elemento primitivo. Para el cifrado los autores proponen el uso de $\widetilde{BPE}_{\tau,\beta}^{-1}(x)$ para la primera función picadillo y $BPE_{\tau,\beta}^{-1}(x)$ para la segunda. Para el descifrado proponen utilizar $BPE_{\tau,\beta}(x)$ para el primer picadillo y $\widetilde{BPE}_{\tau,\beta}(x)$ para el segundo.

TET utiliza una función generadora de números pseudoaleatorios para generar β a partir del afinador T y de la llave K_1 denotada como PRF la cual es una adaptación de OMAC de Iwata y Korasawa [19]. La función PRF se describe en el algoritmo 15.

Algoritmo 15 Función PRF

Entrada: $T_1, ..., T_m$, K y L.
Salida: $PRF_K(T_1, ..., T_m, L)$.

1: $V_0 \leftarrow 0^n$, $X \leftarrow E_K(L)$
2: **para** $i \leftarrow 1$ **hasta** $m-1$ **hacer**
3: $\quad V_i \leftarrow E_K(V_{i-1} \oplus T_i)$
4: **si** $|T_m| = n$ **entonces**
5: $\quad$ **regresar** $E_K(V_{m-1} \oplus T_m \oplus \alpha X)$
6: **si no**
7: $\quad$ **regresar** $E_K(V_{m-1} \oplus T_m \oplus \alpha^2 X)$

En el algoritmo 16 se muestra el cifrado utilizando TET y en el algoritmo 17 se muestra el descifrado.

Respecto al costo computacional de TET: Si se considera un mensaje de m bloques cada uno de tamaño n y que el afinador T es sólo un bloque de tamaño n, el costo computacional para el cifrado es de $2(m+1)$ multiplicaciones en $GF(2^n)$ y $2m+2$ llamados al cifrador por bloques y para el descifrado es de $2m$ multiplicaciones en $GF(2^n)$y $2m+2$ llamados al cifrador por bloques. El esquema requiere cuatro llaves de usuario.

2.6. Comparativo de los Esquemas de Cifrado Entonados

Ahora se dará un análisis comparativo de los *TES* que se implementaron en esta tesis: XCB, HCTR, HCH, EME y TET. Se hace un comparativo del costo computacional para cifrar o descifrar mensajes de m bloques de tamaño n, y tomando en cuenta un afinador T de un solo bloque de tamaño n.

Se introduce el concepto de límite de seguridad, el cual es un número definido como $\sigma_n^w/2^n$ [16] donde σ_n es llamado complejidad de consulta y representa el número de bloques tanto de texto en claro como cifrado que puede ver el adversario, n es la longitud de bloque del cifrador. Entre más pequeño sea el valor de w mayor será la seguridad del esquema. Una descripción más detallada del límite de seguridad puede verse en [11] y para cada *TES* implementado en [8, 17, 34, 31, 15].

En la tabla 2.1 se presenta un comparativo en cuanto al costo computacional de cada *TES* implementado en esta tesis, $[BC]$ indica un llamado al bloque cifrador, $[M]$ es una multiplicación en $GF(2^n)$ y m representa el número de bloques. Es importante destacar que para todos los modos excepto TET el costo computacional de cifrado y del descifrado es el mismo, en TET varía en una multiplicación. El número de pasadas corresponde a cuántas veces se procesa la totalidad de los m bloques.

Algoritmo 16 Cifrado TET

Entrada: $P_1,...,P_m,P_{m+1}$, $T_1,...,T_m$, K_1 y K_2.
Salida: $C_1,...,C_m$.

1: $L \leftarrow len(P)$
2: $i \leftarrow 0^n$
3: $\tau \leftarrow PRF_{K_1}(0,i)$ $\sigma = 1 \oplus \tau^2 \oplus ... \oplus \tau^m$
4: **si** $(\sigma = 0)$ **entonces**
5: $\quad i \leftarrow i+1$
6: $\quad$ **goto 3**
7: $\beta \leftarrow PRF_{K_1}(T_1,...,T_m,L)$
8: $SP \rightarrow 0^n$, $SC \leftarrow 0^n$
9: **para** $i \leftarrow 1$ **hasta** m **hacer**
10: $\quad (SP \leftarrow SP \oplus P_i) \cdot \tau$
11: $SP \leftarrow SP \cdot \sigma^{-1}$
12: **si** $len(P_{m+1}) > 0$ **entonces**
13: $\quad SP \leftarrow SP \oplus pad_{n-r}(P_{m+1})$
14: **para** $i \leftarrow 1$ **hasta** m **hacer**
15: $\quad PP_i \leftarrow P_i \oplus SP$
16: $\quad PPP_i \leftarrow PP_i \oplus \alpha^{i-1}\beta$
17: **para** $i \leftarrow 1$ **hasta** $m-1$ **hacer**
18: $\quad CCC_i \leftarrow E_{K_2}(PPP_i)$
19: **si** $len(P_{m+1}) > 0$ **entonces**
20: $\quad MM \leftarrow E_{K_2}(PPP_{m+1})$
21: $\quad C_{m+1} \leftarrow= P_{m+1} \oplus drop_{n-r}(MM)$
22: **si no**
23: $\quad CCC_{m+1} \leftarrow E_{K_2}(PPP_m)$
24: **para** $i \leftarrow 1$ **hasta** m **hacer**
25: $\quad CC_i \leftarrow CCC_i \oplus \alpha^{i-1}\beta$
26: $\quad SC \leftarrow (SC \oplus CC_i) \cdot \tau$
27: $SC \leftarrow SC \cdot \sigma^{-1}$
28: **si** $len(P_{m+1}) > 0$ **entonces**
29: $\quad SC \leftarrow SC \oplus pad_{n-r}(C_{m+1})$
30: **para** $i \leftarrow 1$ **hasta** m **hacer**
31: $\quad C_i \leftarrow CC_i \oplus SC$
32: **regresar** $C_1,...,C_m,C_{m+1}$

Para una implementación eficiente de algoritmos en hardware, es necesario que sea posible la paralelización de operaciones y el uso de tubería. Por esta razón sólo se implementaron los esquemas presentados en la tabla 2.1, ya que son esquemas total o parcialmente paralelizables y es posible utilizar un cifrador por bloques en tubería.

Algoritmo 17 Descifrado TET

Entrada: $C_1,...,C_m,C_{m+1}$, $T_1,...,T_m$, K_1 y K_2.
Salida: $C_1,...,C_m$.

1: $L \leftarrow len(P)$
2: $i \leftarrow 0^n$
3: $\tau \leftarrow PRF_{K_1}(0,i)$ $\sigma = 1 \oplus \tau^2 \oplus ... \oplus \tau^m$
4: **si** $(\sigma = 0)$ **entonces**
5: $\quad i \leftarrow i+1$
6: $\quad$ **goto 3**
7: $\beta \leftarrow PRF_{K_1}(T_1,...,T_m,L)$
8: $SC \rightarrow 0^n$, $SP \leftarrow 0^n$
9: **para** $i \leftarrow 1$ **hasta** m **hacer**
10: $\quad (SC \leftarrow SC \oplus P_i) \cdot \tau$
11: **si** $len(C_{m+1}) > 0$ **entonces**
12: $\quad SC \leftarrow SC \oplus pad_{n-r}(C_{m+1})$
13: **para** $i \leftarrow 1$ **hasta** m **hacer**
14: $\quad CC_i \leftarrow C_i \oplus SC$
15: $\quad CCC_i \leftarrow CC_i \oplus \alpha^{i-1}\beta$
16: **para** $i \leftarrow 1$ **hasta** $m-1$ **hacer**
17: $\quad PPP_i \leftarrow E_{K_2}(CCC_i)$
18: **si** $len(C_{m+1}) > 0$ **entonces**
19: $\quad MM \leftarrow E_{K_2}(CCC_{m+1})$
20: $\quad P_{m+1} \leftarrow = C_{m+1} \oplus drop_{n-r}(MM)$
21: **si no**
22: $\quad PPP_{m+1} \leftarrow E_{K_2}(CCC_m)$
23: **para** $i \leftarrow 1$ **hasta** m **hacer**
24: $\quad PP_i \leftarrow PPP_i \oplus \alpha^{i-1}\beta$
25: $\quad SP \leftarrow (SP \oplus PP_i) \cdot \tau$
26: **si** $len(C_{m+1}) > 0$ **entonces**
27: $\quad SP \leftarrow SP \oplus pad_{n-r}(P_{m+1})$
28: **para** $i \leftarrow 1$ **hasta** m **hacer**
29: $\quad P_i \leftarrow PP_i \oplus SP$
30: **regresar** $P_1,...,P_m,P_{m+1}$

Modo	**Límite de seguridad**	**Costo computacional**	**Llaves**	**Longitud del mensaje**	**Pasadas**	**Capas de cifrado**	**Paralelizable**
XCB	$\sigma_n^2/2^n$	$(m+6)[BC]+$ $2(m+1)[M]$	1	$[n, 2^{39}]$	2	1	Parcialmente
HCTR	$\sigma_n^3/2^n$	$m[BC]+$ $2(m+1)[M]$	2	$\geq n$	2	1	Parcialmente
HCH	$\sigma_n^2/2^n$	$(m+3)[BC]+$ $2(m-1)[M]$	1	$\geq n$	2	1	Parcialmente
HCHfp	$\sigma_n^2/2^n$	$(m+2)[BC]+$	2	$\geq n$	2	1	Parcialmente
EME	$\sigma_n^2/2^n$	$2(m+2)[BC]$ $2(m-1)[M]$	1	mn $1 \leq m \leq n$	3	2	Sí
TET	$\sigma_n^2/2^n$	$(m+2)[BC]+$ $2(m+1)[M]$ $(m+2)[BC]+$ $2m[M]$	4	$\geq n$ $1 \leq m \leq n$	3	1	Parcialmente

Tabla 2.1: Comparativo de los esquemas de cifrado entonados.

Capítulo 3

Bloques Básicos

Los autores de los *TES* explicados en el capítulo 2 sugieren el uso de un cifrador por bloques que sea seguro, ya que de eso depende la seguridad del esquema en su conjunto. Es necesario utilizar un cifrador por bloques cuya seguridad ya ha sido probada por la comunidad mundial de criptografía, por esa razón se decidió utilizar el Estándar de Cifrado Avanzado AES (por sus siglas en inglés tomadas de *Advanced Encryption Standard*). Los *TES* de las familias Picadillo-Contador-Picadillo y Picadillo-Cifrado-Picadillo, necesitan de un multiplicador para el cálculo de las funciones picadillo. Ya que se requiere un gran número de multiplicaciones, es necesario contar con un multiplicador rápido, para este fin se utilizó el multiplicador de Karatsuba y Ofman. En la primera parte del presente capítulo se explicará en AES y en la segunda el multiplicador de Karatsuba y Ofman.

3.1. AES

En enero de 1997, el Instituto Nacional de Estándares y Tecnología (NIST por sus siglas en ingles) de Estados Unidos anunció una iniciativa para desarrollar un nuevo estándar de cifrado: el Estándar de Cifrado Avanzado (AES por sus siglas en inglés). Éste a su vez se convertiría en un Estándar de Procesamiento de Información del gobierno Federal de Estados Unidos (FIPS por sus siglas en inglés). El ganador reemplazaría al DES (*Data Encryption Standard*). El 2 de octubre de 2000, el NIST anunció el algoritmo ganador: Rijndael, propuesto por los belgas Vincent Rijmen y Joan Daemen. Rijndael es un cifrador de bloque que opera con bloques y llaves de longitudes variables, que pueden ser especificadas independientemente a 128, 192 ó 256 bits [44]. De aquí en lo consecuente se usará AES para referirse al algoritmo Rijndael.

Durante el proceso de selección y después de éste, la seguridad de AES ha sido probada por la comunidad mundial y por esa razón es el idóneo para utilizarse para la implementación de *TES*. Una explicación más amplia de AES y de su seguridad se puede encontrar en [10].

3.1.1. Datos de entrada y salida

Los datos se manejan como vectores de bytes, para el caso que se manejo en esta tesis los datos son de 128 bit, es decir, 16 bytes. Internamente el algoritmo de AES convierte estos vectores en una matriz cuadrada de 4×4 de la siguiente forma:

$$[P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}, P_{12}, P_{13}, P_{14}, P_{15}] \rightarrow \begin{bmatrix} P_0 & P_4 & P_8 & P_{12} \\ P_1 & P_5 & P_9 & P_{13} \\ P_2 & P_6 & P_{10} & P_{14} \\ P_3 & P_7 & P_{11} & P_{15} \end{bmatrix} \quad (3.1)$$

En la ecuación (3.1) se considera a P_0 como el byte más significativo y a P_{15} como el byte menos significativo, este trato se da a la llave secreta K y al vector que se quiere cifrar o descifrar P. A la matriz que se obtiene del vector de entrada se le llama matiz de estado, y sobre ésta se realizan las transformaciones que forman el algoritmo.

3.1.2. Las rondas de AES

El AES es un cifrador por bloques iterativo ya que consiste de la repetida aplicación de una ronda de transformación a la matriz de estado. El número de rondas definidas en el estándar es de 10, este número fue definido por sus diseñadores para el caso de un tamaño de bloque de 128 bits.

El cifrado con el AES comienza con la suma inicial de la llave, denominada *AddRoundKey*, seguida de la aplicación de 9 rondas de transformación y finalmente la ejecución de la *FinalRound*. La ronda inicial y cada una de las rondas siguientes toman como entrada la matriz de estado y una llave de ronda.

La ronda de transformación se compone de 4 pasos: *SubBytes* (SB), *ShiftRows* (SR), *MixColumns* (MC)y *AddRoundKey* (ARK), la ronda final no incluye la aplicación de *MixColumns*. En la figura 3.1 se muestran los pasos del proceso de cifrado de AES.

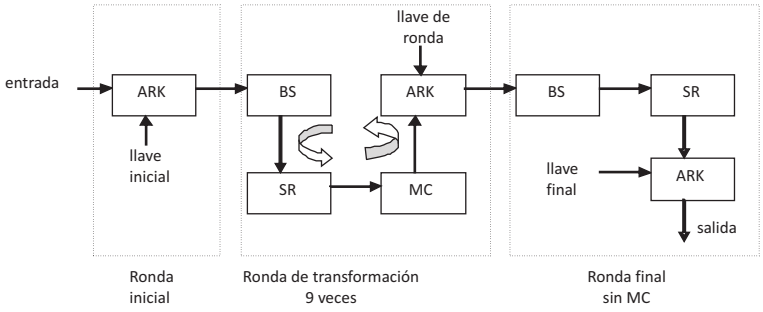


Figura 3.1: Diagrama a bloques de AES.

3.1.3. Transformaciones de AES

Todas las operaciones utilizadas por las transformaciones de AES, emplean aritmética en $GF(2^8)$ utilizando el polinomio irreducible $m(x) = x^8 + x^4 + x^3 + x + 1$ (en lo posterior donde se indique $m(x)$ se hará referencia a dicho polinomio). A continuación se explicarán las transformaciones de AES.

Sustitución de bytes: *Byte Subs* (BS)

Es una transformación no lineal, que consiste en sustituir cada uno de los bytes de la matriz de estados por otro definido en una caja de sustitución llamada *SBOX*. En la figura 3.2 se se ilustra esta transformación. La caja de sustitución contiene 256 valores, es decir, todos los valores representables con 8 bits. La caja de sustitución que utiliza AES se denota como S_{RD}, el diseño de esta se basó en lo siguiente:

1. **No linealidad.**
   - **Correlación:** La máxima correlación entre la entrada y la salida debe tener la menor amplitud posible.
   - **Probabilidad de propagación de diferencias:** La probabilidad de propagación de diferencias debe ser lo más pequeña posible.
2. **Complejidad Algebraica:** La expresión algebraica de S_{RD} en $GF(2^8)$ tiene que ser una expresión lo suficientemente fuerte para resistir ataques de interpolación.

La sustitución de bytes se compone de dos operaciones, el inverso multiplicativo $x = a^{-1} \, mod \, m(x)$ y una transformación afín dada como $y = AF(x) = M \times x \oplus b$ definida en la ecuación (3.2). Para el descifrado se utiliza la transformación afín inversa definida como $x = IAF(y) = M^{-1} \times y \oplus d$ definida en la ecuación (3.3), $a = x^{-1} \, mod \, m(x)$.

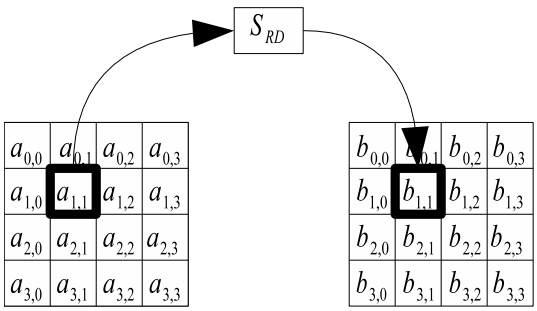


Figura 3.2: Sustitución de bytes de AES.

$$\begin{bmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{3.2}$$

$$\begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \tag{3.3}$$

Corrimiento de filas: *Shift Rows* (SR)

En esta transformación se realiza una transposición donde cada fila del estado es rotada de manera cíclica a la izquierda, una vez para la primera fila, dos veces para la segunda fila y tres veces para la cuarta fila, la primera fila permanece igual como se muestra en la figura 3.3. El criterio seguido para los desplazamientos fue el de difusión óptima, el cual requiere que los cuatro desplazamientos sean diferentes.

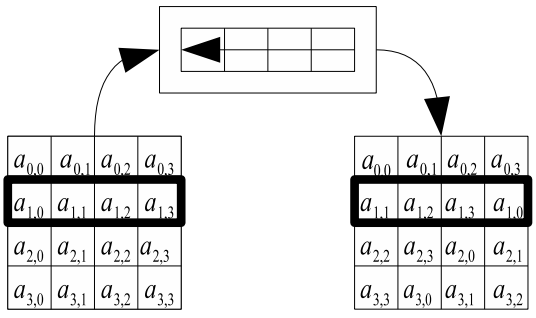


Figura 3.3: Corrimiento de filas de AES.

En el corrimiento de filas inverso (ISR) se realiza el mismo numero de veces el corrimiento, pero a la derecha de cada fila.

Mezclado de columnas: *Mix Column* (MC)

MC es la transformación más costosa de AES ya que utiliza una multiplicación de matrices en $GF(2^8)$. Se toma cada una de las columnas y se multiplica por el polinomio $c(x)\,mod\,x^4+1$ definido

en la ecuación (3.4), en la figura 3.4 se ilustra el efecto de esta transformación. El criterio de diseño para *MixColumns* incluye la dimensión, linealidad, difusión y el rendimiento en un procesador de 8 bits. El criterio de dimensión define la operación de transformación en columnas de 4 bytes.

$$03 \cdot x^3 + 01 \cdot x^2 + 01 \cdot x + 02 \tag{3.4}$$

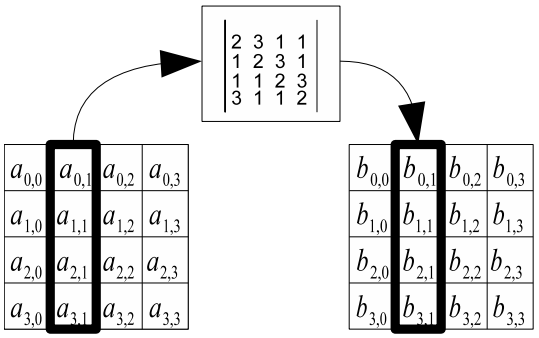


Figura 3.4: Mezclado de columnas de AES.

La multiplicación $a(x) \cdot c(x) \, mod \, x^4 + 1$ se resuelve como se muestra en la ecuación (3.5)

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{3.5}$$

Para la operación inversa (IMC) es necesario multiplicar la inversa de la matriz que aparece en la ecuación (3.5) por cada columna de la matriz de estado, esta operación se muestra en la ecuación (3.6).

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \times \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{3.6}$$

Es posible separar la matriz de la ecuación (3.6) en dos como se muestra en la ecuación 3.7, esto con el propósito de utilizar en el descifrado la matriz utilizada en el cifrado.

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} 05 & 00 & 04 & 00 \\ 00 & 05 & 00 & 04 \\ 04 & 01 & 05 & 00 \\ 00 & 04 & 00 & 05 \end{bmatrix} \tag{3.7}$$

Donde a la matriz,

$$\begin{bmatrix} 05 & 00 & 04 & 00 \\ 00 & 05 & 00 & 04 \\ 04 & 01 & 05 & 00 \\ 00 & 04 & 00 & 05 \end{bmatrix} \tag{3.8}$$

se le llama $ModM$.

Suma de la llave de ronda: AddRoundKey (ARK)

Cada byte de la matriz de estado es combinado con la llave de ronda mediante una operación $\oplus$, cada llave de ronda se deriva de la llave de cifrado usando una generación de llaves que se explica más adelante. El arreglo de llaves de ronda es denominado *ExpandedKey* y es derivado de la llave de inicial. En la ecuación (3.9) se ilustra la operación de suma de la llave de ronda.

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \oplus \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix} \tag{3.9}$$

Generación de llaves de ronda: *Key Schedule* (KS)

La generación de llaves consiste de dos componentes: la expansión de la llave y la selección de la llave de ronda. La expansión de la llave especifica como *ExpandedKey* es derivado de la llave inicial o de cifrado. El número total de bytes en *ExpandedKey* es igual a la longitud del bloque (en bytes) multiplicada por el número de rondas más uno, debido a que el cifrador requiere una llave de ronda para la ronda inicial (ARK), y una para cada una de las rondas siguientes (10 en el caso de AES). Debe notarse que *ExpandedKey* debe ser siempre derivada de la llave de inicial, nunca debe ser especificada de manera directa. El *KS* se escogió basándose en el siguiente criterio:

- **Eficiencia:**
  1. **Memoria a utilizar:** Debe ser posible ejecutar el *KS* utilizando una cantidad pequeña de memoria.
  2. **Rendimiento:** Debe tener un gran rendimiento en una amplia gama de procesadores.
- **Eliminación de la simetría:** Debe utilizar constantes de rondas para eliminar la simetría.
- **Difusión:** Debe tener una difusión eficiente de las diferencias de la llave de cifrado hacia la llave expandida.
- **No linealidad:** Debe mostrar una no linealidad suficiente para no permitir la completa determinación de las diferencias en la llave expandida con la llave de cifrado.

Las llaves de ronda son obtenidas a partir de la expansión de la llave secreta, mediante la adición recursiva de la palabra de 4 bytes $k_i = (k_{0,i}, k_{1,i}, k_{2,i}, k_{3,i})$ a la llave secreta. La llave de cifrado original consiste en 128 bits acomodados como una matriz de bytes de 4×4. Sean $w[0], w[1], w[2]$ y $w[3]$ las cuatro columnas de la llave original. Entonces, estas cuatro columnas son expandidas de manera recursiva para obtener 40 columnas extras como se muestra en la ecuación (3.10).

$$w[i] = \begin{cases} w[i-4] \oplus w[i-1] & \text{si i mod 4 } \neq 0 \\ w[i-4] \oplus T(w[i-1]) & \text{en otro caso} \end{cases} \tag{3.10}$$

Donde $T(w[i-1])$ es una transformación no lineal de $w[i-1]$ calculada como sigue:
Sean w, x, y y z los elementos de la columna $w[i-1]$ entonces:

1. Se realiza un corrimiento cíclico de los elementos para obtener x, y, z y w.

2. Se reemplaza cada byte con el byte correspondiente obtenido de la *S-BOX* $S(x), S(y), S(z), S(w)$.

3. Se calcula la constante de ronda $r_{con}(i) = 02^{(i-4)/4}$ en $GF(2^8)$.

Entonces $T(w[i-1])$ es el vector columna, $(S(x) \oplus r_{con}(i), S(y), S(z), S(w))$. De esta forma, las columnas $w[4]$ hasta $w[43]$ son generadas a partir de las primeras cuatro columnas. La llave de ronda para la i-ésima ronda consiste de las columnas:

$$(w(4i), w(4i+i), w(4i+2), w(4i+3)) \tag{3.11}$$

De esta forma, el cálculo total de la llave $k^{'}$ a partir de la llave k queda como:

$$\begin{array}{cccc} k_0^{'} = k_0 \oplus SBOX(k_{13} \oplus r_{con}) & k_4^{'} = k_4 \oplus k_0^{'} & k_8^{'} = k_8 \oplus k_4^{'} & k_{12}^{'} = k_{12} \oplus k_8^{'} \\ k_1^{'} = k_1 \oplus SBOX(k_{14}) & k_5^{'} = k_5 \oplus k_1^{'} & k_9^{'} = k_9 \oplus k_5^{'} & k_1 3^{'} = k_{13} \oplus k_9^{'} \\ k_2^{'} = k_2 \oplus SBOX(k_{15}) & k_6^{'} = k_6 \oplus k_2^{'} & k_{10}^{'} = k_{10} \oplus k_6^{'} & k_{14}^{'} = k_{14} \oplus k_{10}^{'} \\ k_3^{'} = k_3 \oplus SBOX(k_{12}) & k_7^{'} = k_7 \oplus k_3^{'} & k_{11}^{'} = k_{11} \oplus k_7^{'} & k_{15}^{'} = k_{15} \oplus k_{11}^{'} \end{array} \tag{3.12}$$

SBOX es la aplicación de BS a la matriz de estado de la llave, mientras que r_{con} es una constante de ronda que se calcula en como:

$$r_{con}(i) = 02^{(i-4)/4} \tag{3.13}$$

La operación es calculada en $GF(2^8)$.
Cuando se utiliza AES en para descifrar las llaves se utilizan en orden inverso.

3.1.4. Optimización de las rondas de AES

En la literatura se reportan diferentes formas de optimizar las rondas de AES, en especial resulta importante el caso de *mixcolumn* ya que es la operación más costosa. En [45] los autores proponen una forma de eficientar esta ronda y consiste en lo siguiente, de la ecuación (3.5) se desprende que:

$$\begin{aligned} b_0 &= 2 \cdot a_0 \oplus 3 \cdot a_1 \oplus 1 \cdot a_3 \oplus 1 \cdot a_4 \\ b_0 &= 2 \cdot (a_0 \oplus a_1) \oplus a_1 \oplus a_3 \oplus a_4 \end{aligned} \tag{3.14}$$

De esta manera se reduce únicamente a una multiplicación por 2 y cuatro $\oplus$.

En el caso de su inversa IMC se utiliza la ecuación (3.7), por lo que primero se multiplica la matriz de estado por la matriz *modN* y el resultado se multiplica por la matiz de *mix column*, la razón de esto es la posibilidad de reutilizar para descifrar la matriz de *mix column*. La multiplicación de un vector por *modM* queda como:

$$\begin{aligned} b_0 &= 5 \cdot a_0 \oplus 0 \cdot a_1 \oplus 4 \cdot a_3 \oplus 0 \cdot a_4 \\ b_0 &= 4 \cdot (a_0 \oplus a_3) \oplus a_3 \end{aligned} \tag{3.15}$$

Lo anterior reduce el cálculo de un elemento de IMC a una multiplicación por 4 y dos $\oplus$.

En las ecuaciones 3.5 y 3.6 se observa que para obtener cada uno de los elementos ya sea de MC o IMC, se utilizan las operaciones de las ecuaciones 3.14 y 3.15, para efectos de implementación únicamente es necesario construir un sólo módulo y cambiar los valores a la entrada de este, para obtener el valor adecuado.

En [40] Saqib muestra que es posible optimizar la ecuación (3.12) para paralelizar el cálculo de llaves de ronda, utilizando redundancia en algunos cálculos como se muestra a continuación:

$$\begin{array}{cc} \text{Paso 1} & \text{Paso 2} \\ & k'_4 = k_4 \oplus k'_0 \\ k'_0 = k_0 \oplus SBox(k_{13}) \oplus rcon; & k'_8 = k_8 \oplus k_4 \oplus k'_0; \\ & k'_{12} = k_{12} \oplus k_8 \oplus k_4 \oplus k'_0; \\ & \\ & k'_5 = k_5 \oplus k'_1; \\ k'_1 = k_1 \oplus SBox(k_{14}); & k'_9 = k_9 \oplus k_5 \oplus k'_1; \\ & k'_{13} = k_{13} \oplus k_9 \oplus k_5 \oplus k'_1; \\ & \\ & k'_6 = k_6 \oplus k'_2; \\ k'_2 = k_2 \oplus SBox(k_{15}); & k'_{10} = k_{10} \oplus k_6 \oplus k'_2; \\ & k'_{14} = k_{14} \oplus k_{10} \oplus k_6 \oplus k'_2; \\ & \\ & k'_7 = k_7 \oplus k'_3; \\ k'_3 = k_3 \oplus SBox(k_{12}); & k'_{11} = k_{11} \oplus k_7 \oplus k'_3; \\ & k'_{15} = k_{15} \oplus k_{11} \oplus k_7 \oplus k'_3; \end{array} \tag{3.16}$$

En la ecuación (3.12) existe una dependencia entre los cálculos de las diferentes columnas de matriz de estado de la llave de ronda, en la ecuación (3.16) se eliminan estas dependencias y se realiza el cálculo de la llave de una ronda en sólo dos pasos.

3.2. Multiplicador de Karatsuba y Ofman

La multiplicación utilizando el método tradicional tiene una complejidad de $0(n^2)$, en 1962 Karatsuba y Ofman en [22] presentaron un algoritmo para multiplicación con una complejidad subcuadrática de $GF(n^{log_2(3)})$ [35].

Como ya se ha mencionado, es posible tratar un número en $GF(2^n)$ como un polinomio, así que la multiplicación de dos polinomios puede denotarse como,

$$C(x) = A(x)B(x) = \left(\sum_{i=0}^{n-1}(a_i x^i)\right)\left(\sum_{i=0}^{n-1}(b_i x^i)\right) \tag{3.17}$$

y

$$C^{'}(x) = C(x) \ \ mod \ \ p(x) \tag{3.18}$$

3.2.1. Multiplicadores de Karatsuba y Ofman $2^k m$ bits

Un campo $GF(2^n)$ es construido utilizando un polinomio irreducible $p(x)$ de grado $n = rm$, con $r = 2^k$ y k entero. Sean $A, B \in GF(2^n)$, los dos pueden ser representados polinomialmente como sigue:

$$A = \sum_{i=0}^{n-1}(a_i x^i) = + \sum_{i=\frac{n}{2}}^{n-1}(a_i x^i) + \sum_{i=0}^{\frac{n}{2}-1}(a_i x^i) = x^{\frac{n}{2}} \sum_{i=0}^{\frac{n}{2}-1}(a_{i+\frac{n}{2}} x^i) + \sum_{i=0}^{\frac{n}{2}-1}(a_i x^i) = x^{\frac{n}{2}} B^H + B^L \tag{3.19}$$

y

$$B = \sum_{i=0}^{n-1}(b_i x^i) = + \sum_{i=\frac{n}{2}}^{n-1}(b_i x^i) + \sum_{i=0}^{\frac{n}{2}-1}(b_i x^i) = x^{\frac{n}{2}} \sum_{i=0}^{\frac{n}{2}-1}(b_{i+\frac{n}{2}} x^i) + \sum_{i=0}^{\frac{n}{2}-1}(b_i x^i) = x^{\frac{n}{2}} B^H + B^L \tag{3.20}$$

Usando las ecuaciones 3.19 y 3.20 el producto de polinomios puede definirse como,

$$C = x^m A^H B^H + (A^H B^L + A^L B^H) x^{\frac{m}{2}} + A^L B^L \tag{3.21}$$

El algoritmo Karatsuba Ofman se basa en la idea de que el producto de la ecuación (3.21) puede ser escrito como,

$$C = x^m A^H B^H + A^L B^L + (A^H B^H + A^L B^L + (A^H + A^L)(B^L + B^H)) x^{\frac{m}{2}} = x^m C^H + C_L \tag{3.22}$$

Ahora se define

$$\begin{aligned} M_A &= A^H + A^L \\ M_B &= B^L + B^H \\ M &= M_A M_B \end{aligned} \tag{3.23}$$

Usando la ecuación (3.22), y teniendo en cuenta el hecho de que el producto de polinomios C tiene a los más $2n-1$ coordenadas (en analogía a bits), estas se pueden clasificar como:

$$\begin{aligned} C^H &= [C_{2n-1}, C_{2n-2}, ..., C_{2n+1}, C_n] \\ C^L &= [C_{n-1}, C_{n-2}, ..., C_1, C_0] \end{aligned} \tag{3.24}$$

El costo computacional para calcular el producto C con la ecuación (3.21) es de cuatro productos y tres sumas, mientras que utilizando la ecuación (3.22) es de tres productos y cuatro sumas. Típicamente un producto es más costoso que una suma, con esto puede concluirse que el algoritmo de Karatsuba y Ofman es computacionalmente más simple que el algoritmo clásico. Es posible utilizarlo de manera recursiva para calcular los tres productos que aparecen en la ecuación (3.22) $A^H B^H$, $A^L B^L$ y M. Si se procede recursivamente cada iteración se convertirá en tres multiplicaciones polinomiales reduciéndose el grado de los polinomios en la mitad del valor anterior. En menos de $\lceil log_2(n) \rceil$ iteraciones, se llegará a una multiplicación de un simple entre sólo dos coeficientes (análogamente representa una multiplicación de 2 bits, una *and*), en este punto termina la

recursión. El algoritmo de Karatsuba y Ofman no es muy eficiente para valores de n muy pequeños, por esta razón es conveniente cortar el algoritmo antes de llegar a $n = 1$ y utilizar otros algoritmos para efectuar el siguiente producto como la multiplicación de Mastrovito [42] o la multiplicación tradicional. Para el propósito de esta tesis se parará la recursión en 4 (denotada como $mul_m(A,B)$) como se recomienda en [40]. El algoritmo 18 corresponde al multiplicador de Karatsuba y Ofman $2^k m$.

Algoritmo 18 Multiplicador de Karatsuba y Ofman

Entrada: A y B.
Salida: $C = AB$).

si r=4 **entonces**
  $C \leftarrow mul_m(A,B)$
  regresar
para $i \leftarrow 0$ **hasta** $\frac{r}{2} - 1$ **hacer**
  $M_A i \leftarrow A_i^L + A_i^H$
  $M_B i \leftarrow B_i^L + B_i^H$
$mul2^k(C^L, A^L, B^L)$
$mul2^k(M, M_A, M_B)$
$mul2^k(C^H, A^H, B^H)$
para $i \leftarrow 0$ **hasta** $r - 1$ **hacer**
  $M_i \leftarrow M_i + C_i^L + C_i^H$
para $i \leftarrow 0$ **hasta** $r - 1$ **hacer**
  $M_{\frac{r}{2}+i} \leftarrow C_{\frac{r}{2}+i} M_i$
  Regresar

3.2.2. Reducción modular

En esta tesis se trabaja con multiplicaciones en el campo $GF(2^{128})$, es decir, se utilizan dos operandos de 128 bits, ya que el resultado de la multiplicación tiene a lo más 255 bits, es necesario utilizar un algoritmo de reducción para obtener un resultado de 128 bits. Para el propósito mencionado se utilizó el método de reducción rápida mediante un pentanomio irreducible en el campo [36]. La reducción se realizó con el pentanomio irreducible $p(x) = x^{128} + x^7 + x^2 + x + 1$. El procedimiento se muestra en la figura 3.5. Los bits que cuadros de color gris en la figura 3.5 representan bits que se cancelan al sumarse con sí mismos. Las operaciones necesarias para la reducción se

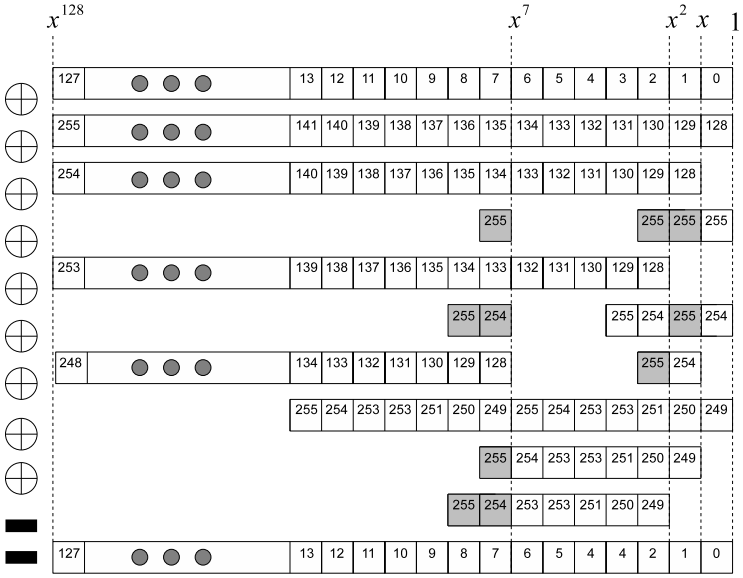


Figura 3.5: Reducción modular $GF(2^{128})$ utilizando el pentanomio irreducible $p(x) = x^{128} + x^7 + x^2 + x + 1$.

muestran a continuación:

$$\begin{aligned}
C'(127...14) &= C(127...14) \oplus C(255...142) \oplus C(254...141) \oplus C(253...140) \oplus \\
&\quad C(248...135) \\
C'(13...9) &= C(13...9) \oplus C(141...137) \oplus C(140...136) \oplus C(139...135) \oplus \\
&\quad C(134...130) \oplus C(255...251) \\
C'(8...7) &= C(8...7) \oplus C(136...135) \oplus C(135...134) \oplus C(134...133) \oplus \\
&= C(129...128) \oplus C(250...249) \\
C'(6...4) &= C(6...4) \oplus C(134...132) \oplus C(133...131) \oplus C(132...130) \oplus \\
&\quad C(255...253) \oplus C(254...252) \oplus C(253...251) \\
C'(3...2) &= C(3...2) \oplus C(131...130) \oplus C(130...129) \oplus C(129...128) \oplus \\
&\quad C(252...251) \oplus C(251...250) \oplus C(250...249) \oplus C(255...254) \\
C'(1) &= C(1) \oplus C(129) \oplus C(128) \oplus C(250) \oplus C(249) \oplus C(254) \\
C'(0) &= C(0) \oplus C(128) \oplus C(249) \oplus C(254) \oplus C(255)
\end{aligned} \tag{3.25}$$

En la ecuación (3.25) se considera que $C'(127...0)$ es una cadena de 128 bits y $C(255...0)$ es una cadena de 256 bits. Como puede observarse este método de reducción se realiza únicamente utilizando operaciones $\oplus$.

Capítulo 4

Implementación de los Bloques Básicos

En este capítulo se describirá la implementación de los bloques básicos necesarios para implementar un *TES*, el cifrador por bloques AES-128 y el multiplicador de Karatsuba y Ofman de 128 bits.

La implementación se realizó utilizando el lenguaje de descripción de hardware VHDL (*Very High Speed Integrated Circuits Hardware Description Language*), y se sintetizó para FPGA (*Field Programmable Gate Array*) específicamente el xc4vlx100-11ff1148 de la famila Vitex4 (ver B) de xilinx [48]. Para la descripción y síntesis se utilizó la herramienta Xilinx ISE 8.1 y para simulación el Model SIM 6.2g.

4.1. Implementación de AES

Como se describió en §3.1.3 para calcular la *SBOX* y su inversa se requiere del inverso multiplicativo en $GF(2^8)$, la transformación afín (AF) y la transformación afín inversa (IAF). Los posibles valores para el inverso multiplicativo en $GF(2^8)$ son 256, estos se precalcularon y se almacenaron en 8 memorias de doble puerto [47]. Para cada byte en la matriz de estado se implementó la sustitución de bytes (BS) y su inversa (IBS) como se muestra en la figura 4.1.

Figura 4.1: Arquitectura para la sustitución de bytes (BS) y su inversa (IBS).

En la arquitectura de la figura 4.1 con el multiplexor se indica qué operación se va a realizar, si la entrada $c/d = 1$ se realiza BS de lo contrario a la salida se tendrá IBS. El circuito para calcular AF se muestra en la figura 4.2 (implementación de la ecuación (3.2)) y el de IAF en la figura 4.3 (implementación de la ecuación (3.3)). Es necesario replicar la arquitectura de la figura 4.1 dieciséis veces, una para cada byte de la matriz de estado.

En §3.1.3 se explicó que la transformación de corrimiento de filas directa e inversa sólo consiste en permutar bytes en la matriz de estado, en hardware esto se reduce a únicamente una reconexión, como se muestra en la figura 4.4 el corrimiento cíclico es a la izquierda y en la figura 4.5 para el

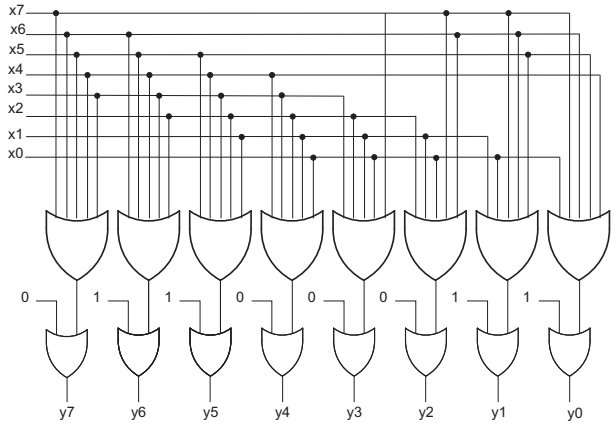


Figura 4.2: Circuito para calcular la transformación afín (AF).

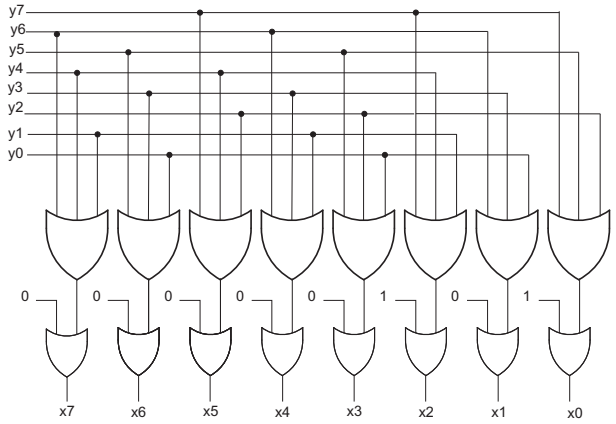


Figura 4.3: Circuito para calcular la transformación afín inversa (IAF).

descifrado el corrimiento es a la derecha. Es importante señalar que esta transformación no significa costo en hardware.

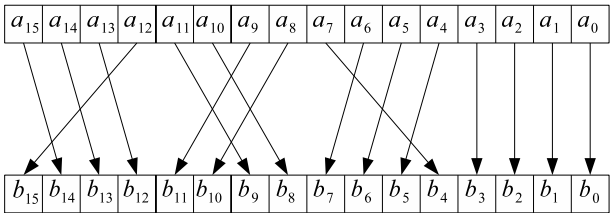


Figura 4.4: Arquitectura para el corrimiento de filas (SR).

El mezclado de columnas (MC) y su inversa (IMC) son las transformaciones más costosas para un diseño en hardware, la razón es que utilizan multiplicaciones de matrices en $GF(2^8)$. Una ventaja representativa es que un operador de la multiplicación es constante, y se pueden realizar multiplicadores específicos.

Tomando como referencia lo expuesto en §3.1.4 se implementó un multiplicador por 2 para MC llamado *xtime*, como se muestra en la figura 4.6 para calcular la ecuación (3.14).

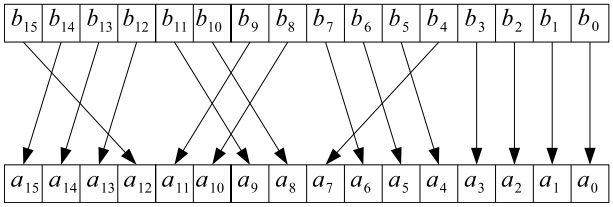


Figura 4.5: Arquitectura para el corrimiento de filas inverso (ISR).

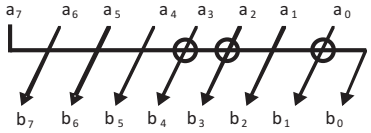


Figura 4.6: *xtime* (multiplicador por 2 en $GF(2^8)$).

El circuito para calcular una columna de la matriz de estado con MC se muestra en la figura 4.7, la caja llamada *xtime* representa el multiplicador de la figura 4.6. Para calcular la totalidad de la matriz se utilizan cuatro circuitos como el mostrado.

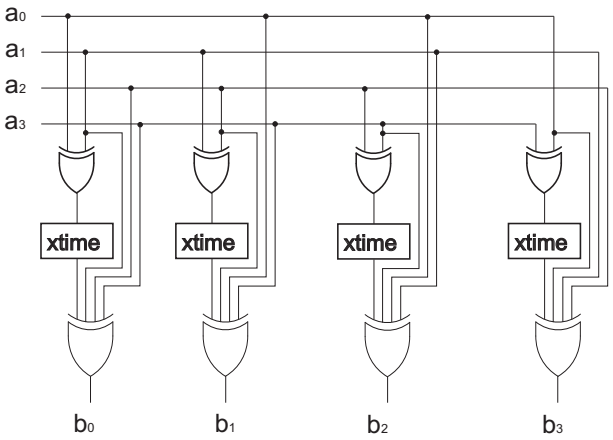


Figura 4.7: Circuito para calcular *mixcolumn*.

En el caso de IMC se implementó un multiplicador por 4, como se muestra en la figura 4.8 llamado *xtime4* para calcular la ecuación (3.15) descrita en §3.1.4. El circuito para multiplicar una columna por la matriz *modM* (§3.1.3) se muestra en la figura 4.9. Al igual que para MC es necesario utilizar cuatro circuitos como el mencionado.

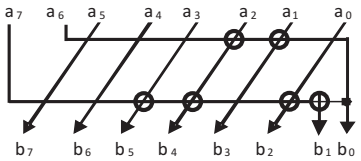


Figura 4.8: *xtime4* (multiplicador por 4 en $GF(2^8)$).

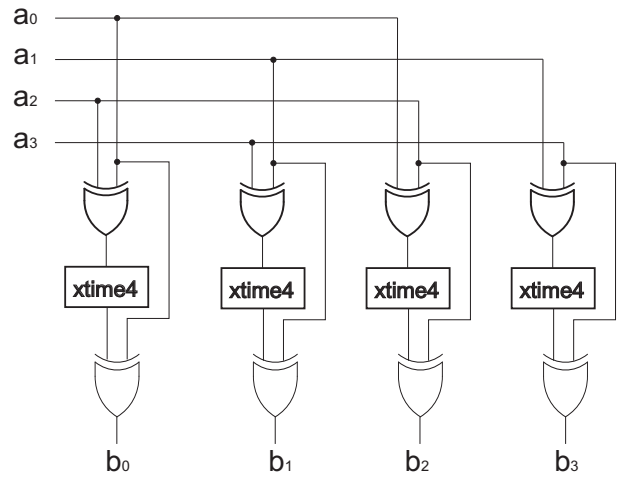


Figura 4.9: Circuito para multiplicar una columna por la matriz *modM*.

Para reutilizar MC en el cálculo de IMC se utilizó la arquitectura mostrada en la figura 4.10, si la entrada $c/d = 0$ del multiplexor MC recibe directamente la matriz de estado y sí $c/d = 1$ MC recibe el producto de *modM* y la matriz de estado.

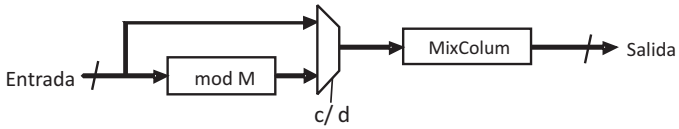


Figura 4.10: Arquitectura para el mezclado de columnas (MC) y su inversa (IMC).

La implementación de una ronda de AES se muestra en la figura 4.12. La entrada c/d indica qué transformaciones deben realizarse mediante los multiplexores, la salida se sincroniza por medio de un *latch*.

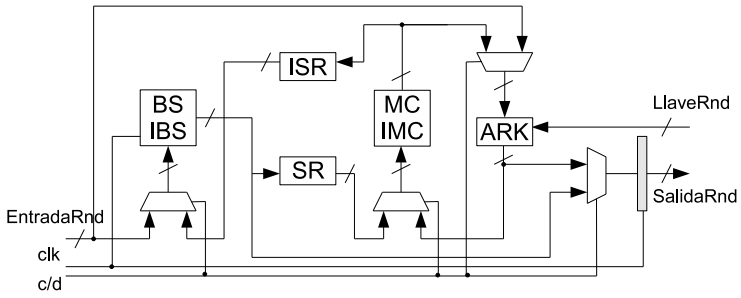


Figura 4.11: Arquitectura para una ronda de AES.

Para la generación de llaves se implementó la ecuación (3.16) mencionada en §3.1.3 que consiste de cinco tablas de consulta, cuatro (*SBOX*) almacenadas en memorias, R_{con} donde se almacenan las constante de ronda y operaciones $\oplus$. En la figura 4.12 se muestra el circuito utilizado para generar una llave de ronda.

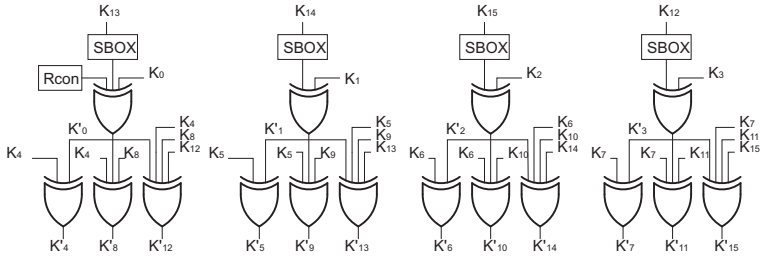


Figura 4.12: Circuito generador de una llave de ronda.

Para generar las 10 llaves de ronda es necesario iterar el circuito de la figura 4.12. Para este propósito se implemento la arquitectura de la figura 4.13. Cuando se hace un reset se toma la llave inicial, de otra forma se toma la entrada de retroalimentación, el circuito es sincronizado a la salida mediante un latch.

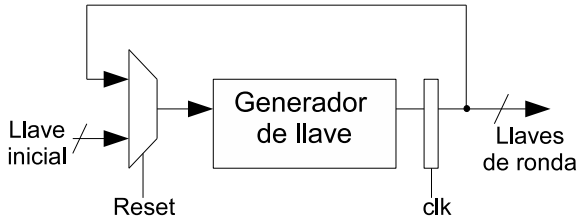


Figura 4.13: Arquitectura del generador de llaves de ronda.

Con el propósito de contar con un AES compacto, se implementó una arquitectura secuencial de AES. En la décima ronda no se realiza el *mix column* (§3.1.2), para lograr un diseño compacto se implementó una sola ronda y se utilizó un multiplexor para distinguir la ronda diez de las demás como se muestra en la figura 4.14. Cuando se realiza un reset al circuito, se comienza el cálculo de las llaves de ronda y se almacenan en una memoria, el circuito espera a que las llaves estén listas para poder cifrar o descifrar. Es necesario calcular primero las llaves ya que como se mencionó en §3.1.3 para descifrar se utilizan en orden inverso. La unidad de control únicamente consta de un contador ascendente y descendente que proporciona el bus de direcciones de la memoria de llaves y el conteo de rondas. Cuando el contador tiene el valor 10 se discrimina la transformación *mix column* y de esta manera calcular la ronda 10. Dependiendo si se cifra o se descifra se utiliza el flujo de información que se mostró en la figura 4.12.

En esta tesis se hizo una evaluación de *TES* utilizados como cifradores de discos, para este propósito se implementó un AES en tubería que a diferencia del secuencial utiliza más recursos de hardware pero la velocidad es significativamente mayor. En la figura 4.15 se muestra la arquitectura. La tubería que se implementó es de 10 niveles, se pueden implementar hasta 70 niveles [39], el número de niveles del *pipe* depende de la cantidad de información que se va a procesar. Para la aplicación de cifrado de discos es suficiente con 10 niveles ya que se cifran/descifran 32 bloques de 128 bits.

En la figura 4.15 se aprecia que las llaves son calculadas y puestas en registros, y cada llave esta directamente conectada a una ronda. A diferencia de la implemetación secuencial, la ronda 10 también se desenrrolló. El flujo de los datos depende de la señal de control c/d ilustrados con flechas indicando la dirección del flujo dependiendo de la operación que se realice cifrado o descifrado.

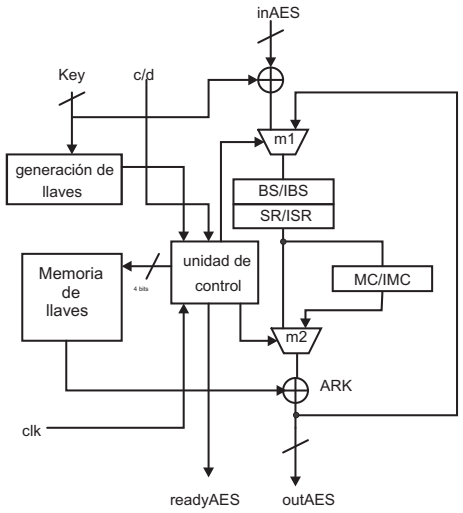


Figura 4.14: Arquitectura secuencial de AES.

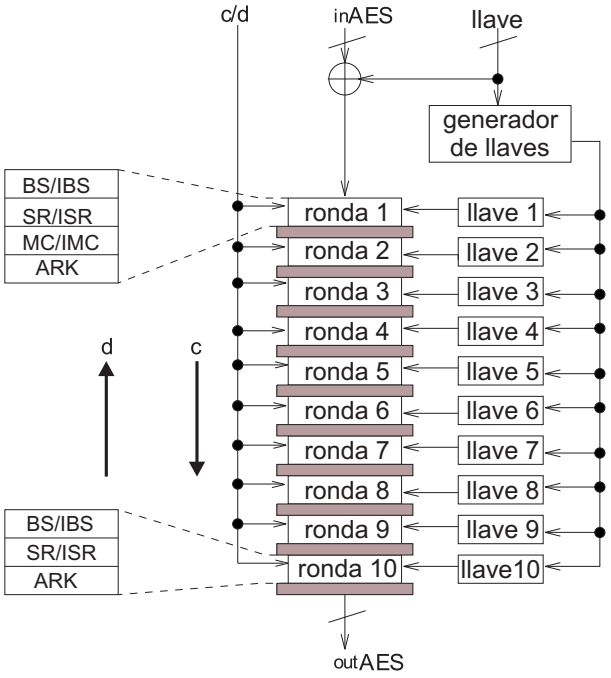


Figura 4.15: Arquitectura en tubería de AES.

Para implementar los *TES* del tipo Picadillo-Contador-Picadillo, se requiere AES tanto para cifrar un bloque (modo simple) y para cifrar un flujo de estos. Para cifrar más de un bloque se utiliza el modo contador, en §2.3 se mencionaron dos tipos el del algoritmo 1 utilizado por XCB y el del algoritmo 6 utilizado en HCTR y HCH. En la figura 4.16 se muestra la arquitectura desarrollada que es capaz de funcionar tanto en modo simple como contador.

En la tabla 4.1 se enlistan las señales de entrada y salida del componente de AES en modo

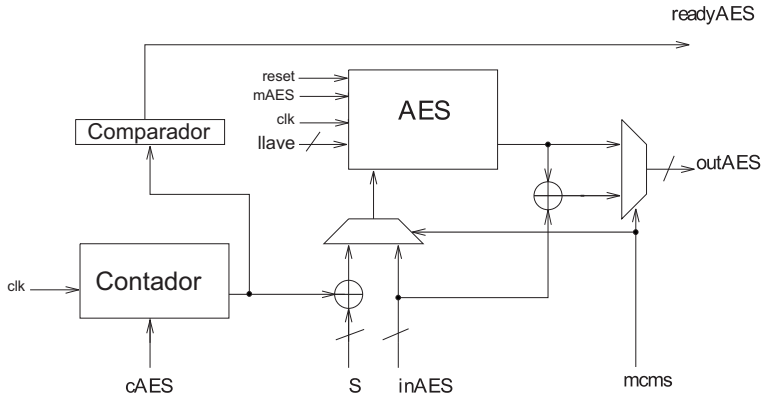


Figura 4.16: Arquitectura de AES en modo simple y modo contador.

Terminal	Función	Bits
inAES	entrada de bloques	128
outAES	salida de bloques cifrados	128
S	vector de inicialización del modo contador	128
llave	llave	128
cAES	reset del contador de rondas	1
mcms	modo contador ó modo simple	1
mAES	cifrado ó descifrado	1
ready	salida válida	1
reset	reset de la generación de llaves	1
clk	reloj	1

Tabla 4.1: Terminales del componente AES en modo simple y modo contador.

simple y contador. La terminal **mcms** indica con el valor de '0'que se cifrará un sólo bloque, y con '1'que se cifrará un flujo de bloques con el modo contador. El **reset** se activa en '1'y reinicializa la generación de llaves. La señal **cAES** reinicializa el contador de rondas y se activa en 1, este contador activa la señal **ready** para indicar que el resultado esta listo. Con la terminal **mAES** se indica si se requiere cifrado ó descifrado (cuando la mcms='1él valor de mAES irrelevante).

4.2. Implementación del multiplicador de Karatsuba y Ofman

Es posible implementar el multiplicador de Karatsuba y Ofman totalmente paralelizado [23] con un circuito totalmente combinatorio, esto permite realizar una multiplicación utilizando únicamente un ciclo de reloj.

El multiplicador de Karatsuba y Ofman se construye de forma recursiva (§3.2), el nivel más bajo de la recursión es un multiplicador tradicional de 4 bits (figura 4.17).

Para formar el multiplicador de 128 bit se utilizan multiplicadores de 4bits (tradicional), de 8 bits, de 16 bits, de 32 bits y de 64 bits. La estructura del multiplicador de 128 bits se muestra en la figura 4.18, cada multiplicador utiliza tres del nivel anterior y se construyen como se muestra en la figura 4.19 donde se ilustra la arquitectura para el multiplicador de 8 bits. Para construir el multiplicador se utilizan tres multiplicadores de 4 bits, el resultado se obtiene a partir de ellos implementando la ecuación (3.22).

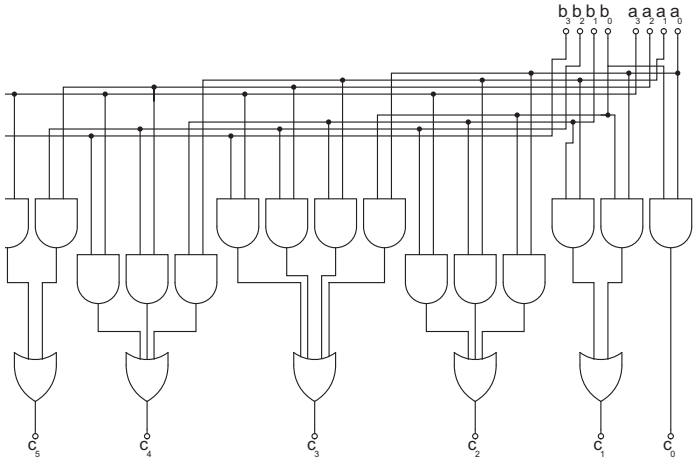


Figura 4.17: Circuito multiplicador tradicional de 4 bits.

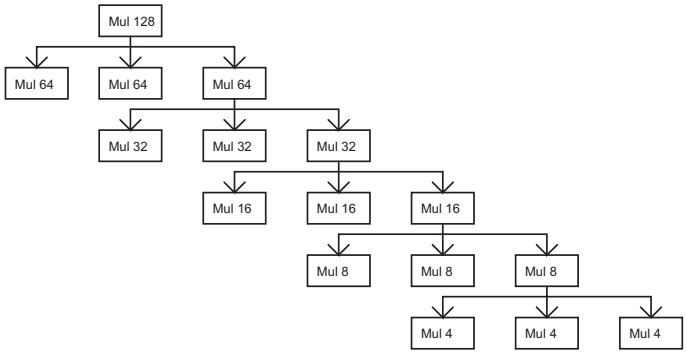


Figura 4.18: Estructura del multiplicador de Karatsuba y Ofman de 128 bits.

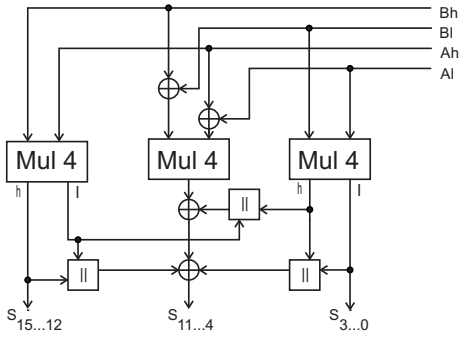


Figura 4.19: Multiplicador de Karatsuba y Ofman para 8 bits.

En los *TES* se utilizan multiplicaciones modulares en $GF(2^{128})$, en el multiplicador diseñado el resultado tiene a lo más 255 bits, por lo anterior es necesario implementar una reducción modular. La reducción modular sólo consiste de la implementación de la ecuación (3.25), que se realoza únicamente con operaciones $\oplus$. El multiplicador modular de 128 bits (mul128) se muestra en la figura 4.20.

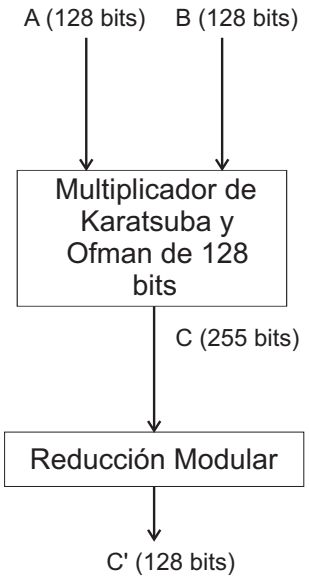


Figura 4.20: Multiplicador de Karatsuba y Ofman para $GF(2^{128})$.

Capítulo 5

Implementación de los Esquemas de Cifrado Entonados

En este capítulo se trata lo referente a la implementación de XCB, HCTR, HCH, EME y TET. Se utilizan los bloques básicos explicados en el capítulo 3 y mostrada su implementación en el capítulo 4.

5.1. Consideraciones diseño

El objetivo específico de esta tesis es evaluar los diferentes *TES* para cifrado de discos. Para poder utilizar los *TES* citados en 2 para cifrado de discos se tomaron las siguientes consideraciones:

- La longitud es fija e igual a 512 bytes que equivalen a 32 bloques de 128 bits.
- El afinador considerado tiene una longitud de 128 bits.
- Los datos de entrada son considerados un sector de un disco y son proporcionados por un circuito independiente, ambos se sincronizan por medio de señales generadas por el circuito de control.
- La validación de los resultados se hizo con vectores de prueba generados con Maple.

5.2. Implementación de XCB

La arquitectura general para XCB se muestra en la figura 5.1, utiliza AES tanto en modo contador como en modo simple y una función picadillo. La sincronización se realiza mediante la unidad de control.

La unidad de control se comunica con el AES por medio de 4 señales todas de un bit: *cAES* para inicializar el contador de rondas, c/d para indicar si se utilizará cifrado o descifrado, *mcms* para indicar si se requiere cifrar un sólo bloque o se cifrará un flujo con el modo contador, con la señal *readyAES* se indica que se tiene una salida válida. El flujo al AES se realiza por medio de tres buses de 128 bits: *inAES* que recibe los bloques que se cifrarán, *outAES* que regresa los bloques cifrados y *S* que recibe el parámetro de inicialización para el modo contador. La comunicación con

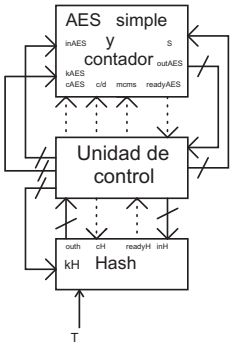


Figura 5.1: Arquitectura general de XCB.

la función picadillo se realiza con 2 señales: *cH* para inicializar el acumulador y el contador de bloques procesados y *readyH* que indica que se ha terminado el cálculo y la salida es válida. El flujo de datos se realiza por los buses inH para la entrada y outH para la salida. A partir de la llave inicial se generan 5 llaves, de éstas la unidad de control selecciona la llave de AES y la de la función picadillo.

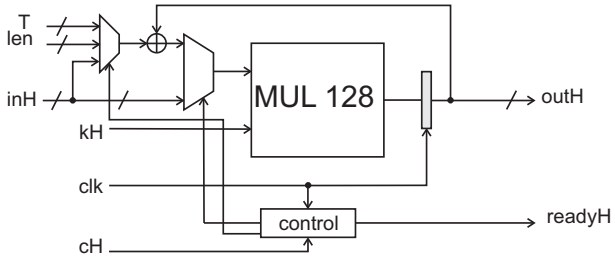


Figura 5.2: Arquitectura de la función picadillo de XCB.

Función picadillo: Consiste de una evaluación polinomial utilizando la regla de Horner, (se mencionó en §2.3.1) en general es la implementación del algoritmo 2 para resolver la ecuación 2.2. La arquitectura de su implementación se muestra en la figura 5.2 y consiste de un multiplicador en $GF(2^{128})$ con una entrada fija y otra que se selecciona según la tabla 5.10. La selección la realiza la unidad de control de la función picadillo mediante un contador de 6 bits.

Valor de contador de control	entrada
1	inH
< 32	$inH \oplus Smul$
32	$T \oplus Smul$
33	$len \oplus Smul$

Tabla 5.1: Selección de entrada de la función picadillo de XCB.

Unidad de Control: Consiste de una máquina de estados que utiliza 14 estados para dar seguimiento a los algoritmos 3 y 4. Los estados son: RESET, SLLAVE1, SLLAVE2, SLLAVE3,

SLLAVE4, HASH1, RAES2, AES1, RASES2, SLLAVE5, SCOUNTER, HASH2, RAES3 y AES3. Se utilizan 8 registros RegK_1, RegK_2, RegK_3, RegK_4 y RegK_5 para almacenar las llaves generadas y regAES, regH y regS para almacenar valores que es necesario preservar durante el algoritmo. Se utiliza un contador de 3 bits para generar los valores necesarios para obtener las 5 llaves (es necesario cifrar de 0 a 4 con la llave inicial). La entrada a AES se selecciona mediante un multiplexor de 4 entradas y una salida, al igual la entrada de la función picadillo se selecciona mediante un multiplexor de 2 entradas y una salida.

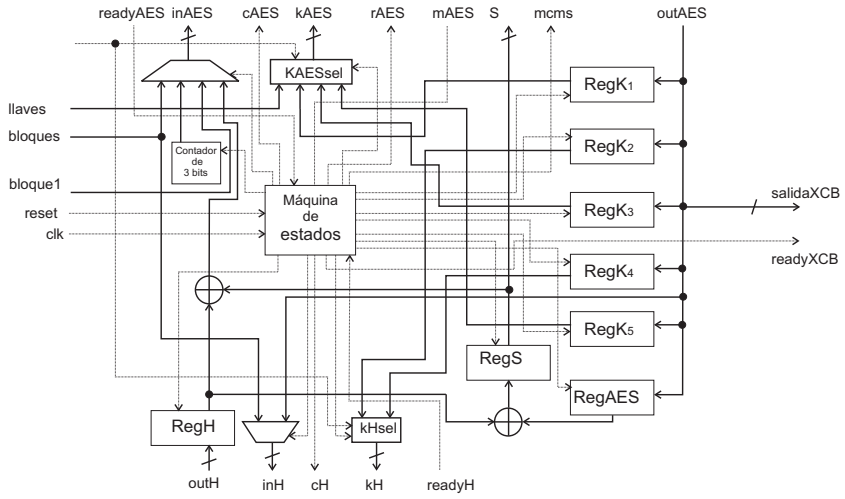


Figura 5.3: Arquitectura de la unidad de control de XCB.

Para la selección de la llave de AES se utiliza kAESsel ya que es necesario seleccionarla de entre cuatro, cuando se descifra se utilizan en orden inverso. El circuito de kAESsel se muestra en la figura 5.4 y la seleccion en la tabla 5.2.

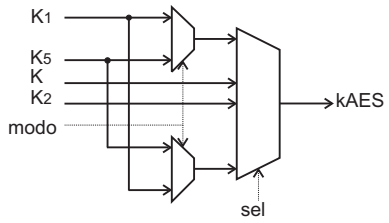


Figura 5.4: Circuito para la selección de la llave de AES.

sel	modo	kAES
0	X	K
1	0	K_1
1	1	K_5
2	X	K_3
3	0	K_5
3	1	K_1

Tabla 5.2: Selección de la llave de AES en XCB.

La llave para la función picadillo es seleccionada de entre dos, para esto se utiliza el módulo kH que se muestra en la figura 5.5 y su selección en la tabla 5.3.

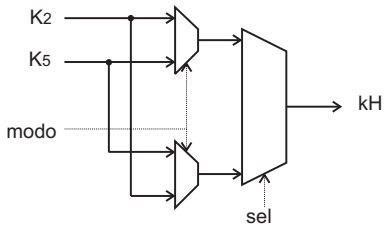


Figura 5.5: Circuito para la selección de la llave de la función picadillo.

sel	modo	kAES
0	0	K_2
0	1	K_4
1	0	K_4
1	1	K_2

Tabla 5.3: Selección de la llave de la función picadillo en XCB.

La máquina de estados para el control de XCB se muestra en la figura 5.6. En el estado RESET se inicializa el circuito y se empieza la generación de llaves, pasa el control a SLLAVE1. En SLLAVE1 se ponen a la entrada de AES en cada ciclo de reloj los valores del contador para obtener las llaves, cuando se activa *readyAES* se obtiene la primera llave que es $E_K(0^{128})$, se almacena en RegK_1 y se pasa el control a SLLAVE2. En SLLAVE 2 se almacena el valor de $E_K(0^{127}||1)$ en RegK_2, se pasa el control a SLLAVE3. En SLLAVE 3 se almacena el valor de $E_K(0^{126}||10)$ en RegK_3, se pasa el control a SLLAVE4. En SLLAVE 4 se almacena el valor de $E_K(0^{126}||11)$ en RegK_4, se pasa el control a SLLAVE5. En SLLAVE 5 se almacena el valor de $E_K(0^{125}||100)$ en RegK_5 y se empieza el cálculo de la primera función picadillo, se pasa el control a RAES1. RAES1 realiza un reset a la generación de llaves de AES para que ahora tome como llave K_1 o K_5 según sea el caso (ver tabla 5.2), pasa el control a AES1. En AES1 hace el cálculo de A, cuando se activa *readyAES* se almacena en regAES y se pasa el control a RAES2.

RAES2 realiza un reset a la generación de llaves de AES para que ahora tome como llave K_2, pasa el control a HASH1. En HASH1 se termina el cálculo del primer picadillo, cuando se activa *readyH*, se calcula $SoutH \oplus regAES$ y se pasa el control a SCOUNTER. En SCOUNTER se realiza el cifrado utilizando el modo contador, cuando se activa *readyAES* se obtiene el primer bloque cifrado C_2, se comienza el cálculo de la segunda función picadillo, pasa el control a HASH2. Mientras se encuentra presente el estado HASH2 se obtienen los bloques cifrados $C_3, ..., C_{32}$ que provienen de AES en modo contador, al activarse la señal *readyH* se obtiene el valor del segundo picadillo, se almacena en regH $S \oplus outH$ y se pasa el control a RAES3. RAES3 realiza un reset a la generación de llaves de AES para que ahora tome como llave K_5 o K_1 según sea el caso (ver tabla 5.2), pasa el control a AES3. Durante AES3 se realiza la última generación de llaves y terminado este se procede al cálculo de $E_K^{-1}(regH)$, al detectarse *readyAES* se obtiene C_1 y con esto se finaliza. En cada estado se genera una palabra de control de 21 bits, su organización se muestra en la tabla 5.4.

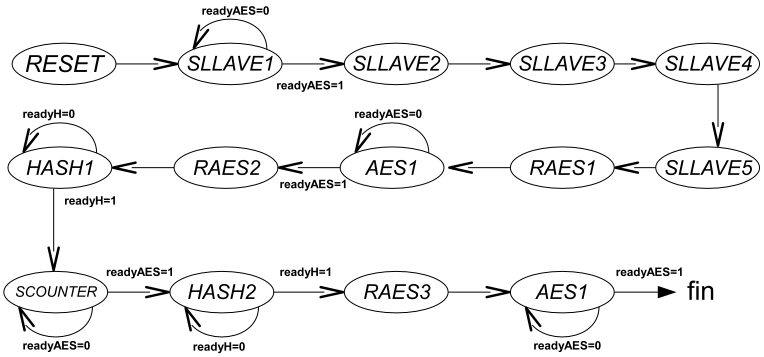


Figura 5.6: Máquina de estados de XCB.

bit de la palabra de control	Función
$cword_0$	Sincronización con los datos de entrada.
$cword_1$	Sincronización con los datos de entrada.
$cword_2$	cH reset del contador de bloques de la función picadillo.
$cword_3$	msmc AES modo simple '0'ó modo contador '1'.
$cword_4$	Modo de trabajo de AES cifrado '0ó descifrado '1'.
$cword_5$	cAES reset del contador de rondas de AES.
$cword_6$	Selección del multiplexor de la entrada de AES.
$cword_7$	Selección del multiplexor de la entrada de AES.
$cword_8$	Reset del contador de 3 bits.
$cword_9$	Activación RegK_1.
$cword_{10}$	Activación RegK_2.
$cword_{11}$	Activación RegK_3.
$cword_{12}$	Activación RegK_4.
$cword_{13}$	Activación RegK_5.
$cword_{14}$	Selección del multiplexor de la entrada de la función picadillo.
$cword_{15}$	Selección del multiplexor de la llave de AES.
$cword_{16}$	Selección del multiplexor de la llave de AES.
$cword_{17}$	Selección del multiplexor de la llave de la función picadillo.
$cword_{18}$	Resete de AES (generación de llaves).
$cword_{19}$	Activación regS.
$cword_{20}$	Activación del regH.

Tabla 5.4: Organización de la palabra de control de XCB.

El comportamiento en el tiempo de XCB se muestra en la figura 5.7, puede observarse que los bloques cifrados $C_2, ..., C_{32}$ se obtienen del ciclo 59 al 88, pero el bloque C_1 es entregado hasta el ciclo 115.

El componente final de XCB se muestra en la figura 5.8.

5.3. Implementación de HCTR

La arquitectura general para HCTR se muestra en la figura 5.9, utiliza AES tanto en modo simple como en modo contador y una función picadillo, ambos son coordinados por la unidad de control para generar una salida válida.

La unidad de control se comunica con el AES por medio de 4 señales todas de un bit: *cAES* para inicializar el contador de rondas, c/d para indicar si se utilizará cifrado o descifrado, *mcms*

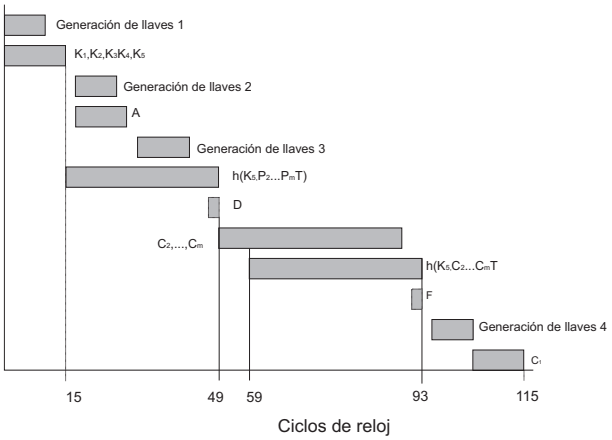


Figura 5.7: Diagrama de tiempo de XCB.

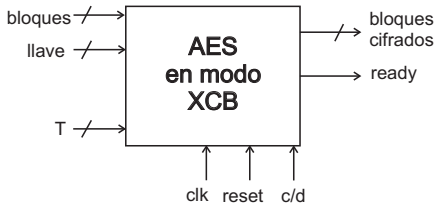


Figura 5.8: Componente final XCB.

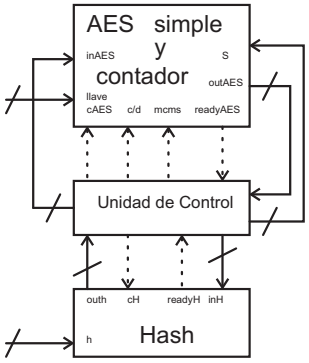


Figura 5.9: Arquitectura general de HCTR

para indicar si se requiere cifrar un sólo bloque o se cifrará un flujo con el modo contador, con la señal *readyAES* se indica que se tiene una salida válida. El flujo al AES se realiza por medio de tres buses de 128 bits: *inAES* que recibe los bloques que se cifrarán, *outAES* que regresa los bloques cifrados y *S* que recibe el parámetro de inicialización para el modo contador. La comunicación con la función picadillo se realiza con 2 señales: *cH* para inicializar el acumulador y el contador de bloques procesados y *readyH* que indica que se ha terminado el cálculo y la salida es válida. El flujo de datos se realiza por los buses inH para la entrada y outH para la salida. Tanto la llave *K* para AES y *h* para la función picadillo provienen de las entradas al sistema y no de la unidad de control.

Función picadillo: La función picadillo para HCTR se explicó en §2.3.2 y consiste de la evaluación del polinomio de la ecuación 2.3 utilizando el algoritmo 5, en general consiste de la aplicación

de la regla de Horner. La arquitectura general de la función picadillo se muestra en la figura 5.10.

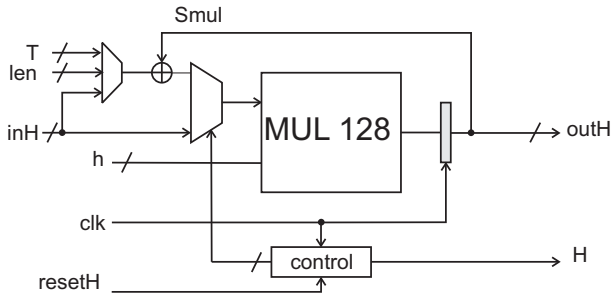


Figura 5.10: Arquitectura de la función picadillo de HCTR.

La función picadillo consiste de un multiplicador en $GF(2^{128})$ con una entrada fija que corresponde a la llave *h*, la otra entrada puede ser tomada de 4 valores posibles como se muestra en la tabla 5.5. La elección de la entrada variable se realiza por medio de la unidad de control de la función picadillo que consiste únicamente de un contador de 6 bits iniciado en 1 que controla el flujo de los bloques y dependiendo de su valor se selecciona dicha entrada.

Valor de contador de control	entrada
1	inH
< 32	$inH \oplus Smul$
32	$T \oplus Smul$
33	$len \oplus Smul$

Tabla 5.5: Selección de entrada de la función picadillo de HCTR.

Unidad de control: Consiste de una máquina de estados que da seguimiento tanto al algoritmo 7 como al algoritmo 8 del modo HCTR mediante 5 estados: RESET, HASH1, AESS, COUNTERAES y HASH2. Dependiendo del estado actual se genera una palabra de control de 8 bits adecuada para efectuar la operación requerida. En el desarrollo del algoritmo es necesario almacenar los valores de *S*, *MM* y *CC*, para esto se recurre al uso de tres registros. La entrada *inH* de la función picadillo puede ser proveniente de la entrada al sistema o de la salida del AES en modo contador, para direccionar dicho bus se utiliza un multiplexor. Tanto los registros como el multiplexor son manejados por la palabra de control generada por la máquina de estados. La entrada al AES es directamente asignado por la máquina de estados. En la figura 5.11 se muestra la arquitectura de la unidad de control para el modo HCTR. Cuando la señal *readyHCTR* es activada indica que los datos presentes en el bus *outHCTR* ya corresponde al cifrado del sector que se proporcionó como entrada.

La máquina de estados se muestra en la figura 5.12, la transición de estados depende de las señales *readyAES* y *readyH*. En RESET se empieza la generación de llaves y se comienza a calcular el primer picadillo, se pasa el control a HASH1. En HASH1 se continúa en paralelo la generación de llaves y el cálculo del primer picadillo hasta que se detecta $readyH = 1$, se calcula *MM* y se almacena en regMM, se pasa el control a AESS. En AESS se calcula $CC = E_K(MM)$, al detectar $readyAES = 1$ se calcula $S = CC \oplus MM$ y se almacena en regCC, *MM* se almacena en regMM y se

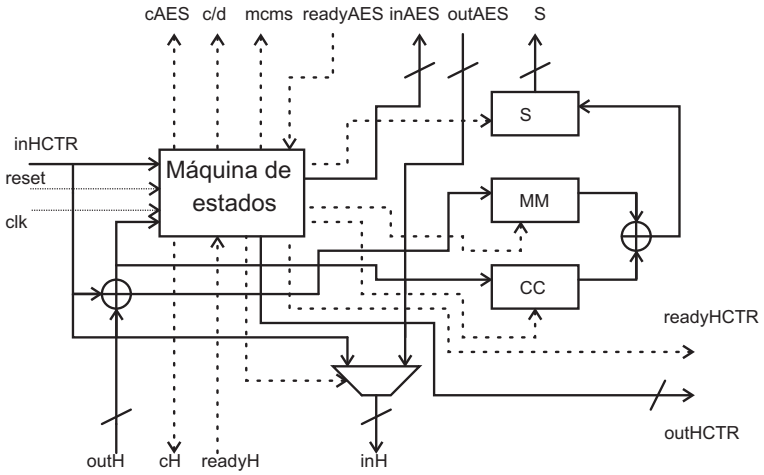


Figura 5.11: Arquitectura de la unidad de control de HCTR

pasa el control a SCOUNTER. En SCOUNTER se comienza a cifrar con AES en modo contador, al detectar $readyAES = 1$ se pasa el cotrol a HASH2. En HASH2 se calculan en paralelo los bloques cifrados $C_2...C_m$ y en paralelo se calcula la segunda función picadillo, al detectar $radyH = 1$ se termina el proceso. La organización de la palabra de control se muestra en la tabla 5.6.

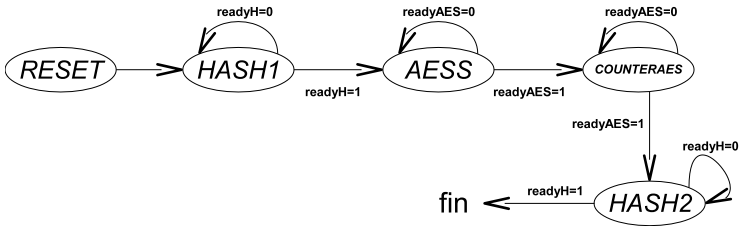


Figura 5.12: Máquina de estados para el control de HCTR

bit de la palabra de control	Función
$cword_0$	Sincronización con los datos de entrada.
$cword_1$	cAES reset del contador de rondas de AES.
$cword_2$	msmc AES modo simple '0'ó modo contador '1'.
$cword_3$	cH reset del contador de bloques de la función picadillo.
$cword_4$	Entrada de la función picadillo '0'bloques de entrada y '1'salida de AES.
$cword_5$	Activación del registro CC.
$cword_6$	Activación del registro MM.
$cword_7$	Activación del registro S.

Tabla 5.6: Organización de la palabra de control de HCTR.

La descripción que se dió de la operación de la máquina de estados es ilustrada en la figura 5.13, donde se observa la paralelización de operaciones y los ciclos de reloj empleados, también se puede observar que se entregan bloques válidos a la salida a partir del ciclo 54 y hasta el 88.

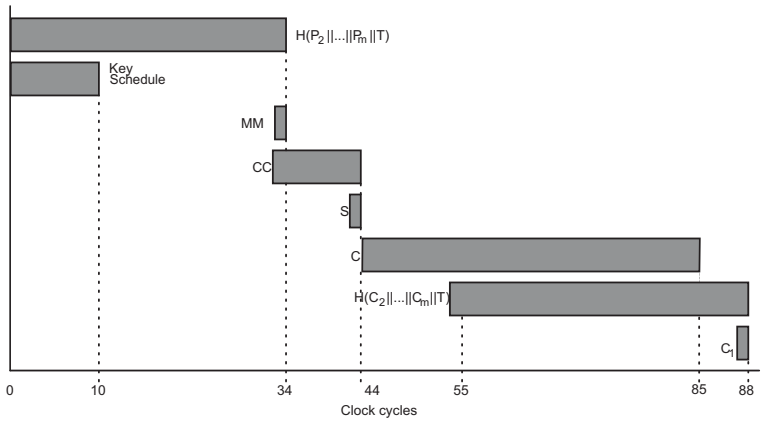


Figura 5.13: Diagrama de tiempo de HCTR

El componente final se muestra en la figura 5.14, puede ser utilizado en diseños más grandes considerándolo como una caja negra.

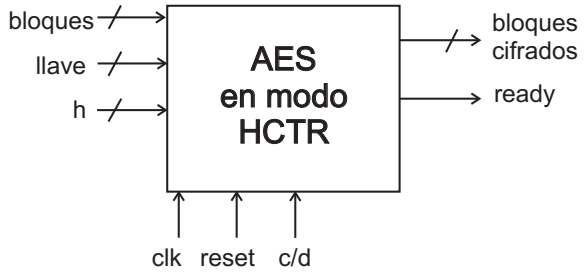


Figura 5.14: Componente final HCTR

5.4. Implementación de HCH

La arquitectura general de HCH que se muestra la figura 5.15 consiste en una función picadillo, AES en modo simple y contador y la unidad de control que coordina el trabajo de ambas para generar una salida válida.

El control de la función picadillo es mediante tres señales: *cH* para inicializar el acumulador y el contador de bloques procesados y *readyH* que indica que se ha terminado el cálculo y la salida es válida. El flujo de datos entre la función picadillo y la unidad de control se reliza con los buses *inH* y $outH$. Los parámetros R y Q son calculados en la unidad de control y enviados mediante buses a la función picadillo. El control de AES es igual que el del modo HCTR.

Función picadillo: Utiliza un multiplicador en $GF(2^{128})$ con una entrada fija y la segunda que puede ser seleccionada de la forma mostrada en la tabla 5.7, la selección se hace dependiendo del valor de la señal *reset*. En la figura 5.16 se muestra la arquitectura de la función picadillo, la señal *readyH* indica que se ha terminado de calcular el picadillo y el resultado se encuentra disponible en $outH$.

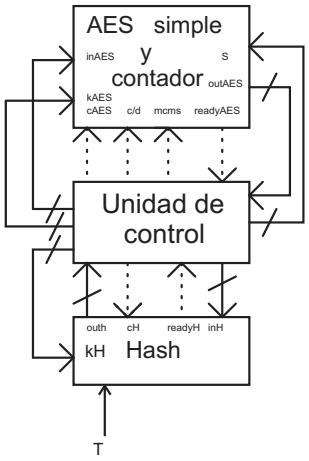


Figura 5.15: Arquitectura de la unidad de control de HCH

reset	entrada
1	*inH*
0	$inH \oplus Smul.$

Tabla 5.7: Selección de entrada de la función picadillo de HCH.

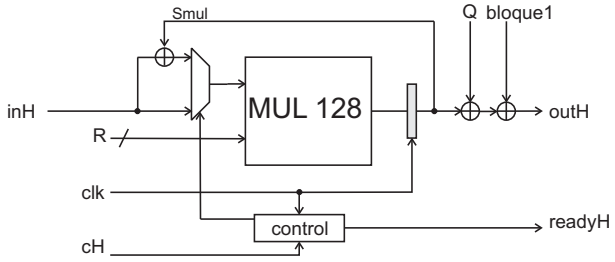


Figura 5.16: Arquitectura de la función picadillo de HCH.

Unidad de control: Se compone de una máquina de estados que da seguimiento a los algoritmos 10 y 11 mediante 8 estados: RESET, AES1, AES2, HASH1, AES3, AES4,ECOUNTER y HASH2. En el transcurso es necesario almacenar datos, para ello es necesario utilizar 6 registros: R, Q, S, I, U_1 y M_1. Se utilizan dos multiplexores, uno para la entrada a la función picadillo *inHCH* y otro para Q para seleccionar entre Q y xQ. Tanto el multiplexor como los registros son controlados utilizando bits de la palabra de control que genera la máquina de estados. Para el cálculo de xQ se utiliza un multiplicador por 2 $GF(2^{128})$ (se ilustra su diseños en la figura 5.17), en lo posterior se hara referencia a este multiplicador como $xtime128$. La entrada al AES es asignada por la máquina de estados. La arquitectura general del control de HCH se muestra en la figura 5.18.

En RESET se inicia la generación de llaves y se asigna T a *inAES*, pasa el control a AES1. En AES1 se calcula $R = E_k(T)$ en paralelo con la generación de llaves y cuando se activa *readyAES* se almacena outAES en regR, reinicializa el contador de rondas, calcula $R \oplus len$ y pasa el control a AES2. En AES2 se calcula $Q = E_k(R \oplus len)$ y en paralelo se comienza a calcular la función picadillo M_1, cuando *readyAES* se activa reinicializa el contador de rondas, almacena Q en regQ, se calcula xQ y pasa el control a HASH1. En HASH1 se termina de calcular el picadillo M_1, cuando se activa

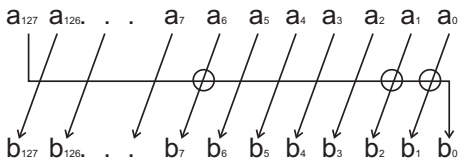


Figura 5.17: Multiplicador por 2 en $GF(2^{128})$ ($xtime128$).

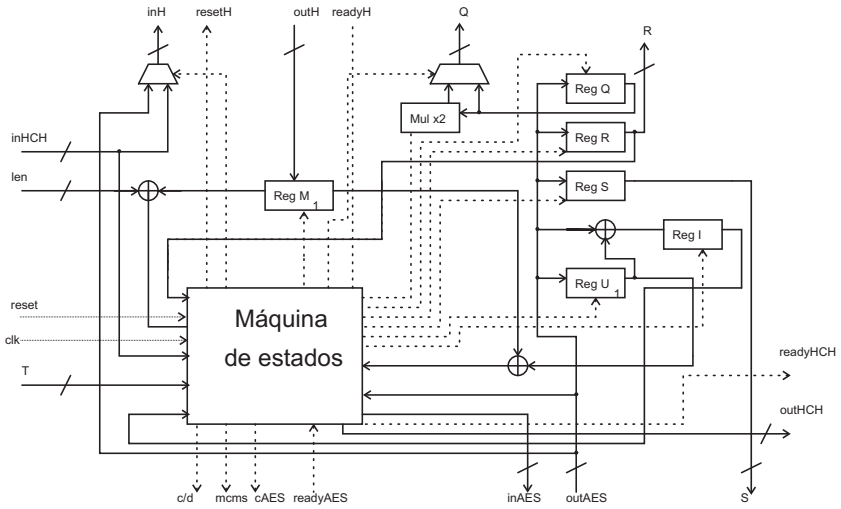


Figura 5.18: Arquitectura de la unidad de control de HCH

la señal *readyH* se almacena el resultado de la función picadillo en $regM_1$ y pasa el control a AES3. En el estado AES3 se calcula $U_1 = E_k(M_1)$, al detectar que se activa la señal *readyAES* se calcula $I = M_1 \oplus U_1$, se almacena en RegI y pasa al estado AES4. En AES4 se calcula $S = E_k(I)$, cuando se activa *readyAES* se almacena en RegS y se pasa al estado ECOUNTER. En el estado ECOUNTER se comienza la fase de cifrado con AES en modo contador, cuando detecta que se activa la señal *readyAES* se activa la función picadillo y se pasa al estado HASH2. Durante el estado HASH2 se realiza en paralelo el cifrado $C_2, ..., C_m$ y el cálculo de la función picadillo, al activarse la señal *readyH* se tiene el resultado de la función picadillo que corresponde a C_1. La máquina de estados se muestra en la figura 5.19 y la organización de la palabra de control en la tabla 5.8.

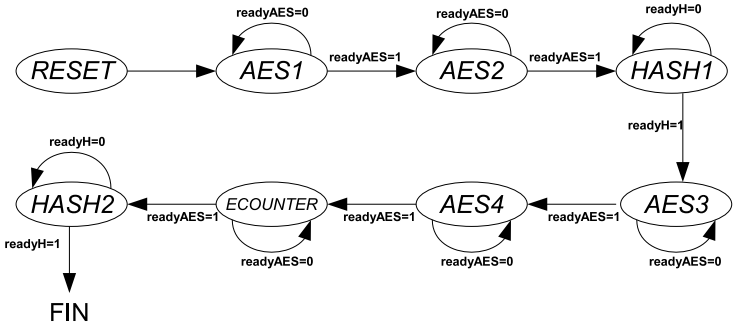


Figura 5.19: Máquina de estados de HCH

bit de la palabra de control	Función
$cword_0$	Sincronización con los datos de entrada.
$cword_1$	Entrada de la función picadillo *inH* '0'bloques de entrada y '1'salida de AES *outAES*.
$cword_2$	Reset de la función picadillo *cH*.
$cword_3$	Sincronización con los datos de entrada.
$cword_4$	Reset del contador de rondas cAES.
$cword_5$	Modo de operación de AES simple '0'ó contador '1', *msmc*.
$cword_6$	Indica que el modo contador esta listo para cifrar un flujo de datos.
$cword_7$	Carga registro *R*.
$cword_8$	Carga registro *Q*.
$cword_9$	Carga registro U_1.
$cword_{10}$	Carga registro M_1.
$cword_{11}$	Carga registro *S*.
$cword_{12}$	Indica si AES trabaja en modo cifrado o el indicado por la terminal *modo*.
$cword_{13}$	Carga registro I.

Tabla 5.8: Organización de la palabra de control de HCH.

Las operaciones mencionadas en la descripción de la máquina de estados se ilustran en la figura 5.20, y se muestra que el diseño implementado de HCH permite cifrar el sector de un disco en 106 ciclos de reloj, empezando a entregar los bloques válidos en el ciclo 74.

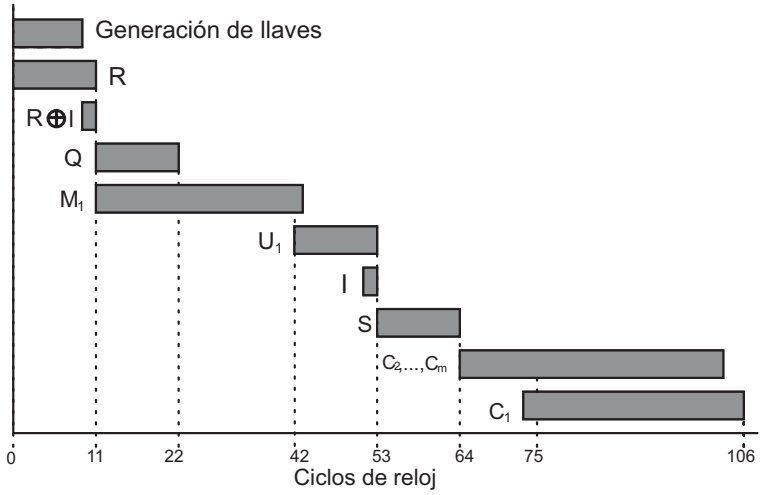


Figura 5.20: Diagrama de tiempo de HCH

El flujo de control es prácticamente el mismo en el cifrado y en el descifrado, sólo existen dos variantes. Una es usar Q o xQ tanto en el primer picadillo como en el segundo esto se resuelve utilizando un mutiplexor en la entrada Q de la función picadillo el cual selecciona Q cuando el $estado = HASH1$ y $modo = 0$, de otro modo selecciona xQ y para el segundo picadillo selecciona xQ sí $estado = HASH2$ y $modo = 0$ de otro modo selecciona Q. Otra modificación que debe hacerse es en $estado = AES3$ ya que hay que calcular $U_1 = E_k^{-1}(M_1)$.

La construcción final de HCH se muestra en la figura 5.21.

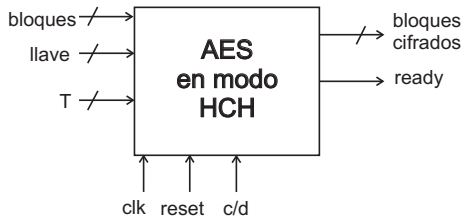


Figura 5.21: Componente final HCH.

5.4.1. HCHfp (HCH para mensajes de longitud fija)

La modificación hecha a HCH por sus autores para adaptarlo específicamente para cifrado de discos, en el aspecto de la implementación redujo los requerimientos en un registro y un estado en el control que significaron 11 ciclos de reloj menos que el esquema original. El diagrama de tiempo para HCHfp se muestra en la figura 5.22.

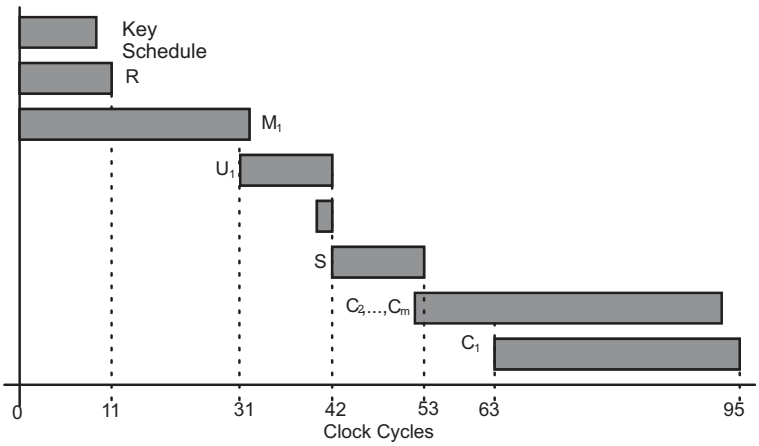


Figura 5.22: Diagrama de tiempo de HCHfp.

Ya que el esquema HCHfp no depende de la longitud no calcula el parámetro $Q = R \oplus len$, únicamente calcula R y el polinomio lo evalúa sobre α que es una entrada al componente final HCHfp como se muestra en la figura 5.23.

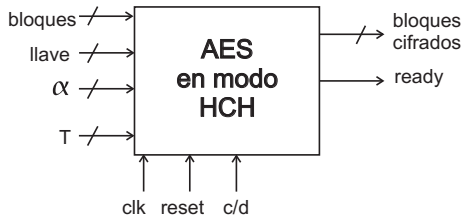


Figura 5.23: Componente final HCHfp.

5.5. Implementación de EME

La arquitectura general para EME, consta solamente de AES en modo simple y la unidad de control (figura 5.24).

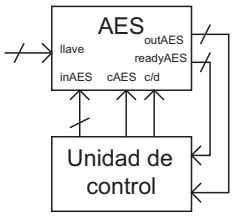


Figura 5.24: Arquitectura de la unidad de control de EME

El esquema EME no utiliza multiplicaciones, pero utiliza otras operaciones para realizar el enmascaramiento. Se requiere de un componente que sea capaz de generar la secuencia $L, 2L, 4L, 8L...$, se utiliza el módulo llamado *xtime*128*b* (figura 5.17) y se hace iterar con la arquitectura de la figura 5.25, en lo posterior para hacer referencia a este módulo se usará *xntimes*. Se construyo también un módulo para calcular una $\oplus$ acumulada, se muestra en la figura 5.26, y se hará referencia a ella como *ACC*.

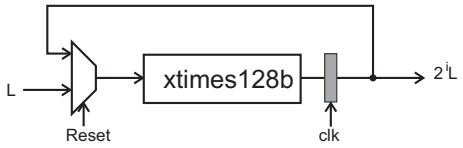


Figura 5.25: Arquitectura para calcular 2^iL (*xntimes*).

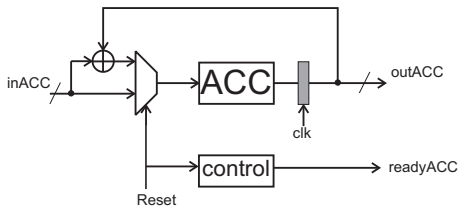


Figura 5.26: Arquitectura para calcular una $\oplus$ acumulada (*ACC*).

El circuito de control para EME se muestra en la figura 5.27, el bloque principal es una máquina de 11 estados: RESET, AES1, XTIME1, AES2, XTIME2, ACC1, AESI, XTIME3, AES3, ACC2 y AES4. En cada estado se genera una palabra de control de 14 bits de 14 bits con los que se controlan los diferentes componentes, se utilizan 5 registros L, PPP_1, MP, MC y M, un modulo *xtime* y dos *ACC*. Observando los algoritmos 13 y 14 se observa que es necesario almacenar los valores $PPP_1, ..., PPP_{32}$, para este propósito se utiliza una memoria RAM de 128×32 generada con *Core Generator* de Xilinx [46] y un contador de 5 bits para generar el bus de direcciones de la memoria.

La máquina de estados se muestra en la figura 5.28. En el estado RESET se empieza la generación de llaves y se pasa el control a AES1. Durante el estado AES1 se calcula $L' = E_K(0^n)$,

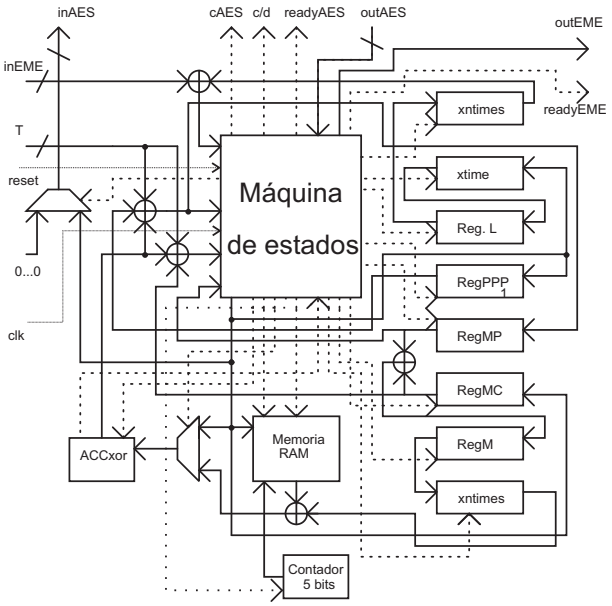


Figura 5.27: Arquitectura de la unidad de control de EME.

cuando *readyAES* se activa se calcula $L = 2L'$ y se almacena en RegL, se pasa el control a XTIME1 donde se comienza el cifrado ECB con $PP_1 = E_K(L \oplus P_1)$, se pasa el control a AES2. En AES2 se continúa con el cifrado con ECB solo que ahora se calcula $PP_i = 2^{i-1}L \oplus P_i$ y $PPP_i = E_K(PP_i)$ donde $2^{i-1}L$ se calcula con el módulo *xntimes* (figura 5.25) e $i = 2...32$, cuando se activa *readyAES* se comienza a calcular *SP* utilizando el módulo *ACC* (figura 5.26), se pasa el control XTIME2. En XTIME2 se generan las señales necesarias para poder almacenar en la memoria RAM los valores $PPP2,...,PPP_32$ que provienen de la salida de AES, se pasa el control a ACC1. Durante ACC1 se procede con el almacenamiento de la salida de AES y se continúa con el cálculo de *SP*, al activarse *readyACC* se calcula $MP = SP \oplus T$, se almacena en *regMP* y se pasa el control a AESI.

Durante el estado AESI se calcula $MC = E_K(MP)$, cuando se detecta *readyAES* activa sa almacena *MC* en regMC, se calcula $M = MP \oplus MC$, se sincroniza la memoria RAM para leer los datos que se almacenaron en ella y se pasa el control a XTIME3. En XTIME3 se empieza a calcular $CCC_2 = 2M \oplus PPP_2$ y *SC*, PPP_2 proviene de la memoria RAM, $2M$ se calcula con un módulo *xntimes* y *SP* con un módulo ACC, en paralelo se comienza con el cálculo de $CC_2 = E_K(CCC_2)$ y se pasa el control a AES3. Durante AES3 se continúa el cálculo de $CCC_i = 2^{i-1}M \oplus PPP_i$ y *SC* con $i = 3...13$, al activarse *readyAES* indica que el primer dato cifrado ECB esta listo y se empieza el cálculo de $C_2 = CC_i \oplus 2L$ $2L$ se calcula con el módulo *xtimes* utilizado anteriormente con L y se pasa el control a ACC2. En ACC2 se termina de calcular CCC_i, CC_i y *SC* y se continua el cálculo de C_i, al activarse *readyACC* se calcula $CCC_i = SC \oplus T$ y se envía al AES para calcular $E_K(CCC_1)$ y se pasa el control a AES4. El estado AES4 es el último de la máquina de estados y en este se termina de calcular *Ci* acabando con la salida C_32 y cuando se activa *readyAES* se obtienen el valor de C_i. La descripción de la máquina de estados puede ser comprendida de forma más clara observando la figura 5.29 donde se muestra el diagrama de tiempo y la paralelización de operaciones. La palabra de control de EME es de 14 bits que se organizan como se muestra en la tabla 5.9.

bit de la palabra de control	Función
$cword_0$	Sincronización con los datos de entrada.
$cword_1$	Entrada de la función picadillo *inH* '0'bloques de entrada y '1'salida de AES *outAES*.
$cword_2$	Reset del contador de rondas cAES Reset de la función picadillo *cH*.
$cword_3$	Reset de *xntimesL*.
$cword_4$	AES en modo cifrado '1'ó en el modo presente en la señal externa *modo* '0'.
$cword_5$	Reset de ACC.
$cword_6$	Reset del contador para la memoria RAM.
$cword_7$	Carga registro PPP_i.
$cword_8$	Carga registro *L*.
$cword_9$	Modo de la memoria RAM '1éscritura y '0'lectura.
$cword_{10}$	Carga registro *MP*.
$cword_{11}$	Reset de *xntimesL*.
$cword_{12}$	Carga registro *MC*.
$cword_{13}$	Carga registro *M*.

Tabla 5.9: Organización de la palabra de control de EME.

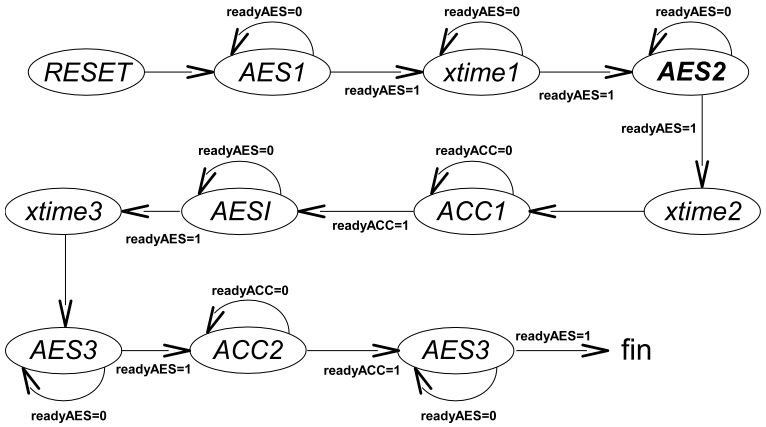


Figura 5.28: Máquina de estados de EME.

Para el descifrado únicamente se realiza el cambio de $E_K()$ por $E_K^{-1}()$ en todos los llamados al cifrador, excepto en el primero donde se calcula L'.

Finalmente el componente final de XCB se muestra en la figura 5.30.

5.6. Implementación de TET

La arquitectura general de TET se muestra en la figura 5.31, utiliza AES únicamente en modo simple además de una función picadillo.

La comunicación entre la unidad de control y AES se hace de la misma forma que en EME la única variante es que en TET se utilizan dos llaves diferentes con AES. La función picadillo tiene una forma especial, utiliza la evaluación polinomial acompañada a la salida de esta de un bloque *xntimes* (figura 5.25). La arquitectura para la evaluación polinomial se muestra en la figura 5.32

El multiplexor por donde entran los bloques se controla con la señal de *cH*, si es '1éntra el primer bloque ó si es '0'la entrada se toma de la retroalimentación de la misma forma que en el picadillo de HCH. La otra entrada al multiplicador también puede variar entre dos valores, para la selección se utiliza un multiplexor cuya selección es controlada mediante las condiciones de la tabla

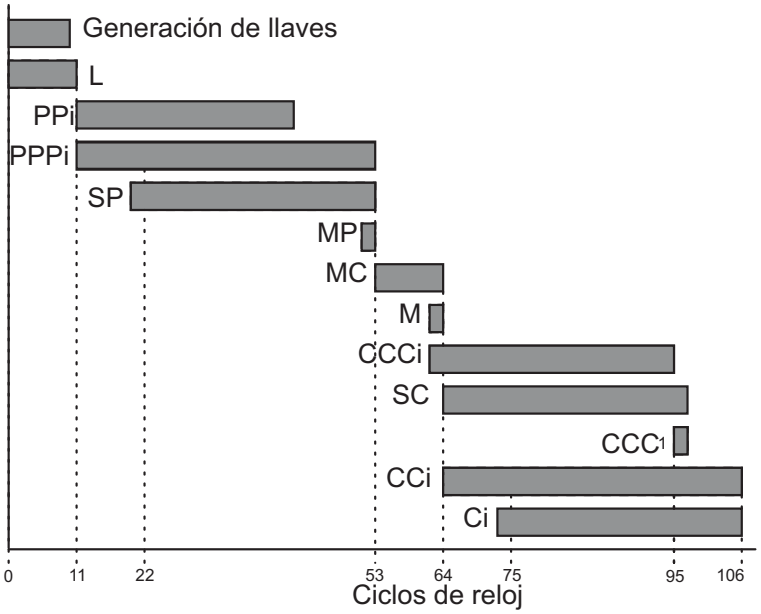


Figura 5.29: Diagrama de tiempo de EME.

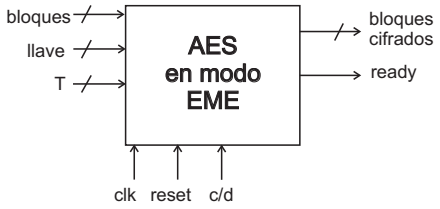


Figura 5.30: Componente final EME.

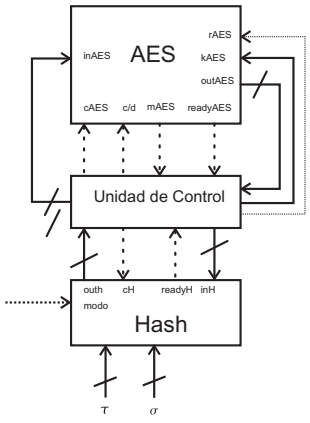


Figura 5.31: Arquitectura general de TET.

5.10.

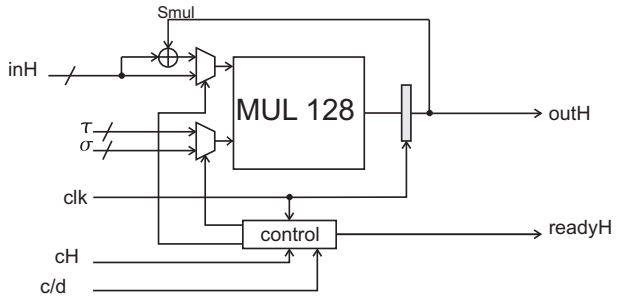


Figura 5.32: Arquitectura de la función picadillo de TET.

Contador	entrada
< 31	τ
31	σ.

Tabla 5.10: Selección de entrada de la función picadillo de TET.

Unidad de control: Consiste de una máquina de estados, 2 registros Regβ y RegH, dos módulos *xntimes* (figura 5.25), un módulo *xtime* (figura 5.17), una memoria RAM de 128×32 bits y un contador de 6 bits para el bus de direcciones de la memoria. La entrada a AES se selecciona mediante un multiplexor de 3 entradas una salida, la llave de igual forma se selecciona mediante un multiplexor de 2 entradas una salida al igual que la entrada de la función picadillo. La memoria RAM es utilizada ya que es necesario almacenar los valores $CC_1, ...CC_{32}$ (ver algoritmo 16). El control se realiza mediante 10 estados: RESET, PRF1, PRF2, RSTKSCH, HASH1, ECB1, SINCRO, HASH2, PLS4 y SALDATOS. Cada estado genera una palabra de control de 14 bits, su organización se muestra en la tabla 5.11.

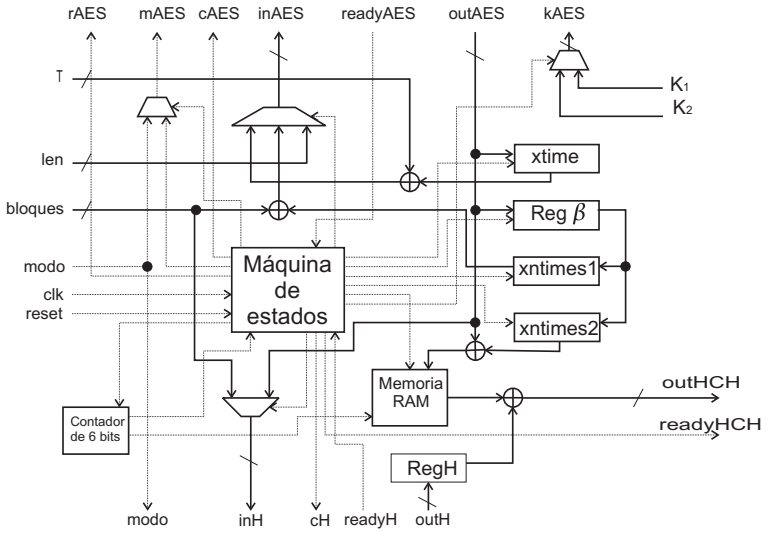


Figura 5.33: Arquitectura de la unidad de control de TET.

bit de la palabra de control	Función
$cword_0$	Sincronización con los datos de entrada.
$cword_1$	Reset de la función picadillo cH.
$cword_2$	Entrada del picadillo *inH* '0'bloques de entrada y '1'salida de AES *outAES*.
$cword_3$	Reset de *xntimes*1 opera con β.
$cword_4$	AES modo cifrado '1'ó el modo presente en el puerto *modo* '0'.
$cword_5$	Reset del contador de rondas cAES.
$cword_6$	Multiplexor de llaves para AES.
$cword_7$	Selección del multiplexor en la entrada de AES.
$cword_8$	Selección del multiplexor en la entrada de AES.
$cword_9$	Activación de Regβ.
$cword_{10}$	Reset de *xntimes*1 opera con β.
$cword_{11}$	Reset del contador de la memoria RAM.
$cword_{12}$	Modo de la memoria RAM '1éscritura y '0'lectura.
$cword_{13}$	Reset AES (Generación de llaves).

Tabla 5.11: Organización de la palabra de control de TET.

La máquina de estados para el control de TET se muestra en la figura 5.34. En el estado RESET se incializa el sistema y se comienza con la generación de llaves utilizando K_1, pasa el control a PRF1. En PRF1 se empieza el cálculo β con la función $PRF(T,L)$ (algoritmo 15) haciendo $X = E_{K_1}(0^{128})$, cuando se activa la señal *readyAES* se calcula $T \oplus 2X$ donde $2X$ se calcula con un módulo *xtime* y se pasa el control a PRF2. En PRF2 se continúa el cálculo de PRF ahora se calcula $PRF = (T,L) = E_{K_1}(T \oplus 2X \text{ donde } 2X)$, al activarse la señal *readyAES* el cálculo se termino y se almacena en Regβ, se pasa el control a RSTKSCH. RSTKSCH efectúa un reset al AES que inicializa la generación de llaves pero ahora toma K_2, también se inicia el cálculo de la evaluación polinomial de la primera función picadillo y pasa el control a HASH1. Durante el estado HASH1 se termina la generación de llaves y se continua la evaluación polinomial, al activarse *readyH* se almacena el valor de la evaluación polinomial en regH y pasa el control a ECB1.

Durante el estado ECB1 se completa el cálculo del picadillo con la ayuda del modulo *xntimes*1 para generar los valores $PP_i = P_i \oplus SP$ y $PPP_i = PP_i \oplus 2^{i-1}\beta$ donde SP es el valor almacenado en RegH y $i = 1...32$, al activarse la señal *readyAES* se tiene el primer bloque cifrado con AES en modo ECB, se almacena en la memoria RAM, se comienza la evaluación polinomial de la segunda función picadillo y se pasa el control al estado HASH2. En HASH2 se calcula en paralelo el cifrado $CCC_i = E_{K_2}$ y la evaluación polinomial, al detectar *readyH* activa se almacena el resultado de la evaluación polinomial en regH y se pasa el control a PLS4. EL estado PLS4 prepara la memoria RAM para ser leída, prepara el módulo *xtime*2 y pasa el control a SALDATOS. El estado SALDATOS entrega los bloques cifrados efectuando el cálculo $CC_i = CCC_i \oplus 2^{i-1}\beta$ y $C_i = CC_i \oplus SC$ donde CCC_i provienen de la memoria RAM y SC de regH, cuando se detecta el bit 5 del contador (bit(5)) en '1'se han entregado los 32 bloques cifrados y se finaliza el proceso.

En la figura 5.35 se muestra el diagrama de tiempo la implementación de TET, en él se aprecia de forma clara que operaciones se realizan en paralelo. La arquitectura propuesta entrega el primer bloque cifrado en el ciclo 78 y finaliza en el 110. El componente final de TET se muestra en la figura 5.36.

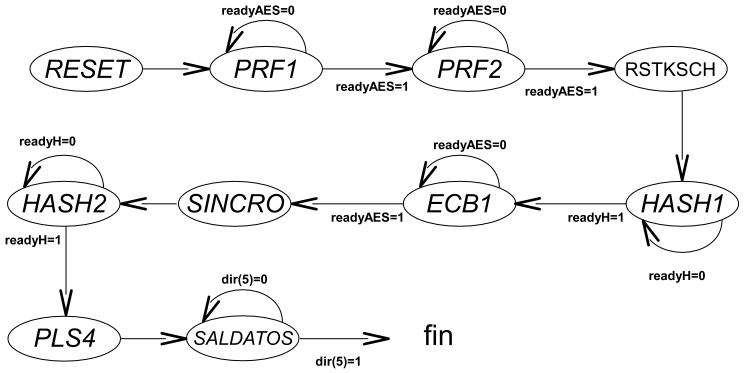


Figura 5.34: Máquina de estados de TET.

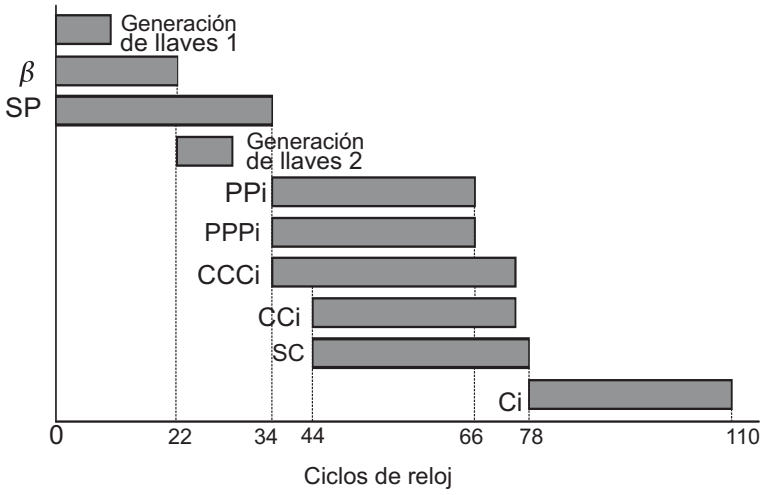


Figura 5.35: Diagrama de tiempo de TET.

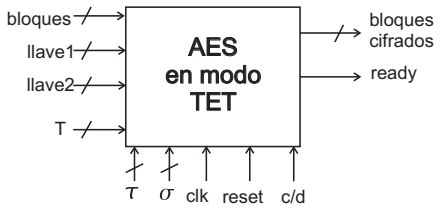


Figura 5.36: Componente final TET.

Capítulo 6

Resultados y Análisis

En este capítulo se muestran los resultados obtenidos y se realiza un análisis de estos. En la primera parte mostrarán las métricas utilizadas para evaluar un diseño en FPGA, en la segunda parte utilizando dichas métricas se evaluarán los resultados obtenidos de las implementaciones mostradas en §5 y finalmente se realiza una comparación especulativa con soluciones existentes en software.

6.1. Medidas de rendimiento en FPGA

Para poder cuantificar la calidad de un diseño en un FPGA se utilizan dos criterios: El área utilizada y la velocidad de procesamiento. En [40] se definen como sigue:

Área: Se define como la cantidad de recursos ocupados por un diseño y se mide en *slices* utilizados. Algunos FPGAs tienen otros recursos como BRAMs, multiplicadores, etc (ver B). Si se ocupa alguno de estos recursos es importante reportarlos ya que no ocupan *slices*, de esta manera se justifica el ahorro de área de un diseño.

Tiempo de procesamiento: Para evaluar la respuesta en el tiempo de los diseños realizados en esta tesis se utilizarán tres criterios. El primero es el tasa de cifrado que se refiere a la cantidad de bits que se procesan en un segundo y se define como:

$$\text{tasa de cifrado} = \frac{\text{máxima frecuencia} \cdot \text{número de bits de salida}}{\text{ciclos de reloj utilizados}} (bits/seg) \tag{6.1}$$

El segundo criterio es el tiempo total utilizado para cifrar 32 bloques de 128 bits, que es el tamaño del sector de un disco. El tercer criterio es la latencia que se refiere al tiempo que tarda un diseño en entregar el primer bloque válido en la salida.

Relación tiempo área: Este parámetro indica una relación entre los dos anteriores se denota TPA (por sus siglas en inglés, se toman de *throughput per area*). Lo que se busca es que un diseño tenga una velocidad máxima ocupando un espacio mínimo. Cuando se utilizan BRAMs del FPGA se calcula como:

$$TPA = [(slices + 128 \cdot BRAMs) \cdot \text{tiempo total}]^{-1} \tag{6.2}$$

A continuación con las métricas explicadas se evaluarán los diseños implementados en esta tesis.

6.2. Resultados

En la tabla 6.1 se muestran los dos bloques principales utilizados en la implementación de los TES: Una ronda de AES utilizando cifrado y descifrado (c/d) y el multiplicador. La frecuencia de operación es definida por dicha la ruta crítica que se refiere al máximo número de compuertas por las que debe pasar una señal desde la entrada a la salida del circuito. Una ronda de AES c/d utiliza aproximadamente un 30 % del área que utiliza el multiplicador, en lo que respecta a la ruta crítica la de la ronda de AES c/d es mayor en aproximadamente un 10 % que la correspondiente a la ruta crítica del multiplicador. La diferencia de área entre la ronda del AES c/d y el AES solo cifrado se debe a la forma de implementar la transformación de sustitución de bytes (ver §4.1), al ser implementado para c/d se requiere de las transformación afín y su inversa además de los inversos multiplicativos en $GF(2^8)$ almacenados en una memoria. Cuando sólo se emplea el cifrado es posible utilizar únicamente la memoria y ahorrar los slices de la transformacón afín.

Diseño	Slices	B-RAM	Ruta crítica (nS)
Ronda de AES c/d en tubería	1215	8	10.998
Ronda AES solo cifrado en tubería	298	8	6.71
Multiplicador	3223	0	9.85

Tabla 6.1: Resultados de la implementación de una ronda de AES y del multiplicador de Karatsuba y Ofman.

Utilizando los tres módulos descritos anteriormente se obtuvieron los resultados de la tabla 6.2. La implementación de AES c/d utilizando tubería es más lenta que la implementación de la función picadillo. La implementación de AES solo cifrado utilizando tubería es más rápida que la función picadillo. Con lo anterior puede decirse que cuando se utiliza el AES c/d en tubería la ruta crítica del TES será definida por este, en el caso de utilizar AES sólo cifrado en tubería la ruta crítica es definida por la función picadillo. El dato que aparece en la tabla 6.2 como función picadillo se refiere específicamente a la función picadillo de HCH, ya que es la de mayor ruta crítica. El análisis anterior es válido para los modos que utilizan función picadillo: XCB, HCTR, HCH, HCHfp y TET. En el caso de EME la ruta crítica es definida básicamente por el AES utilizado.

Técnica	Slices	B-RAM	Frecuencia (MHz)	Ciclos de reloj	Tasa de cifrado (Mbps)	TPA
AES secuencial c/d	1301	18	81.967	10	1.05	2273.7
AES tubería c/d	6368	85	83.88	1	10.74	4863.17
AES tubería solo cifrado	2468	85	149	1	19.07	11162.7
Función picadillo	4014	-	101.45	1	13.00	25247.0

Tabla 6.2: Resultados de las implementaciones de los bloques básicos.

En la tabla 6.3 se muestran los recursos de hardware requeridos por cada modo de operación implementado. Se consideran el número de entradas que tiene AES, las BRAMs adicionales a las utilizadas por AES, el número de multiplicadores *xtime* (multiplicadores por 2 en $GF(2^8)$, ver §5.4), cuántos registros de 128 bits se requieren y la cantidad y tamaño de los multiplexores utilizados. La

cantidad de recursos utilizados por un modo de operación influyen en la complejidad de diseño, lo que directamente se ve reflejado en la ruta crítica.

Modo	Multiplexor inAES	B-RAM Adicionales	Multiplicador *xtime*	Registros 128 bits	Multiplexor 2×128	Multiplexor 3×128
HCTR	3×128	-	-	3	2	1
HCHfp	4×128	-	1	5	5	-
HCH	5×128	-	1	6	5	-
EME	4×128	2	3	5	5	-
TET	3×128	2	3	2	4	-
XCB	4×128	-	-	8	6	2

Tabla 6.3: Recursos de hardware utilizados por los seis TES implementados.

En la tabla 6.4 se muestran los resultados experimentales para la implementación de los seis modos de operación utilizando un AES c/d en tubería. El número de ciclos es mayor en uno al reportado para cada modo en §5, ya que se utiliza un ciclo de reloj para el reset inicial. La ruta crítica de todos los modos es definida por el AES c/d, ya que como se mostró en la tabla 6.1 tiene una mayor ruta crítica que la función picadillo. En términos de área se observa que EME es el modo más económico ya que no utiliza función picadillo. La ruta crítica de EME es dada por el AES c/d más tres sumas en serie, por esta razón la frecuencia de EME es parecida a la de los demás modos de operación. En términos de área utilizada los más costosos son XCB y HCH. El modo de operación más rápido es HCTR seguido de EME, HCHfp, HCH, TET y XCB respectivamente. El mejor TPA fue obtenido por HCTR ya que su velocidad es la máxima y la cantidad de área que ocupa es razonable.

Modo	**Slices**	**B-RAM**	**Frecuencia (MHz)**	**Ciclos de reloj**	**time (μS)**	**Latencia (μS)**	**Tasa de cifrado (Gbps)**	**TPA**
HCH	13662	85	65.939	107	1.623	1.137	2.524	26.06
HCHfp	12970	85	66.5	96	1.443	0.992	2.84	29.04
HCTR	12068	85	79.65	89	1.117	0.703	3.67	39.01
XCB	13418	85	54.021	116	2.147	1.110	1.91	19.168
EME	10120	87	67.835	107	1.577	1.120	2.60	29.82
TET	12072	87	60.505	111	1.834	1.305	2.23	23.494

Tabla 6.4: Implementaciones utilizando AES c/d en tubería con generación de llaves

En la tabla 6.5 se muestran los seis modos de operación utilizado un AES secuencial. En este escenario el más lento es EME ya que requiere dos pasadas de cifrado para 32 bloques cada una, y cada bloque requiere de 11 ciclos de reloj. El cálculo de la función picadillo no es afectado debido a que el multiplicador es un circuito combinatorio que entrega el resultado en un solo ciclo de reloj. En este escenario HCTR continúa siendo el más rápido seguido de HCHfp, HCH, TET, XCB y EME.

Los modos de operación HCTR, HCH, HCHfp, TET y XCB cuando se utilizan en modo cifrado solo utilizan llamadas a AES en modo cifrado. También se realizaron los cinco modos de operación mencionados utilizando un AES en tubería solo cifrado. En este escenario el más rápido y con mayor TPA es EME que casi llega a la velocidad del AES (ver tabla 6.2). Después de EME lejos en cuestión de velocidad le siguen HCTR, HCHfp y HCH. TET resulta ser el más ineficiente y más lento, indicado por su TPA. Estos resultados se muestran en la tabla 6.6.

El número de ciclos de reloj reportado en la tabla 6.6 utilizados por HCHfp varía en dos ciclos a lo reportado en la tabla 6.4, esto se debe a una optimización realizada en la función picadillo de

Modo	Slices	B-RAM	Frecuencia (MHz)	Ciclos de reloj	Tiempo (μS)	Tasa de cifrado (Gbps)	TPA
HCH	8688	18	64.026	416	6.497	0.631	14.00
HCHfp	7903	18	64.587	405	6.270	0.653	15.73
HCTR	7006	18	77.737	388	4.991	0.82	21.53
XCB	8351	18	52.108	455	8.731	0.469	10.749
EME	5053	20	65.922	716	10.861	0.377	12.09
TET	7030	20	58.592	421	7.185	0.570	14.512

Tabla 6.5: Implementaciones utilizando AES secuencial con generación de llaves

Modo	Slices	B-RAM	Frecuencia) (MHz)	Ciclos de reloj	time (μS)	latencia μS	Tasa de cifrado (Gbps)	TPA
HCHfp	7513	85	95.801	98	1.022	0.678	4.00	53.198
HCTR	6996	85	98.580	89	0.902	0.557	4.54	62.018
EME	4200	87	149.09	107	0.717	0.496	5.71	90.942
TET	7165	87	93.035	111	1.193	0.849	3.43	45.802

Tabla 6.6: Implementaciones utilizando AES en tubería solo cifrado.

HCHfp que permitió elevar la frecuencia a cambio de elevar en dos el número de ciclos de reloj necesarios para cifrar un sector de un sector de un disco.

6.3. Comparación especulativa con soluciones en software

De acuerdo a la tabla 2.1 (ver §2.6) el modo EME requiere de $2(m+1)$ llamadas a AES más algunas operaciones extras. En las implementaciones realizadas se asumió que el mensaje tiene una longitud fija igual a 32 bloque de 128 bits que equivalen a 512 bytes. Por esta razón , el costo computacional en software de EME es un poco más del costo de 66 llamadas al cifrador por bloques. Usando estas estimaciones en la tabla 6.7 se compara la implementación en hardware reconfigurable hecha en esta tesis contra las mejores implementaciones en software reportadas en la literatura abierta. Usando la técnica *bit-slice*, la implementación más rápida de AES solo cifrado fue reportada recientemente en [28]. Este diseño requiere sobre 147.2 ciclos de reloj para cifrar un bloque de 128 bits utilizando un procesador Intel Core 2 Duo a 2.135 GHz. Otras implementaciones competitivas en software son utilizadas en la tabla 6.7. Como se muestra en la tabla 6.7 el diseño de EME presentado en esta tesis supera a la mejor implementación en software presentada por un factor de aceleración de al menos 6.34.

Diseño	Procesador	Ciclos por sector	Tiempo EME	Aceleración
EME usando AES en [28]	Intel Core 2 Duo	9715.2	4.55	6.34
EME usando AES en [27]	AMD Atlhon 64	11120	4.675	6.52
EME usando AES en [25]	AMD Atlhon 64	13134	5.47	7.63
EME usando AES en [3]	AMD Atlhon 64	14150	5.896	8.717
EME en esta tesis	FPGA Virtex 4	107	0.717	1

Tabla 6.7: Comparativo especulativo del EME implementado con las mejores soluciones existentes en hardware.

Capítulo 7

Conclusiones

En este capítulo se dan las conclusiones obtenidas durante el desarrollo de esta tesis. Primero se enlistan los resultados alcanzados, después se exponen las conclusiones del trabajo realizado analizando los resultados mencionados y por último se menciona el trabajo futuro que puede realizarse para mejorar este trabajo.

7.1. Resultados obtenidos

En la realización de esta tesis, se obtuvieron los siguientes resultados:

- Se realizaron tres implementaciones de AES, en modo secuencial c/d, en tubería c/d y solo cifrado en tubería. Ambas implementaciones en tubería utilizan 10 rondas desdobladas. Dichas implementaciones son competitivas con las encontradas en la literatura abierta.
- Se implementó un multiplicador Karatsuba Ofman totalmente paralelizado eficiente para $GF(2^{128})$.
- Se obtuvieron implementaciones eficientes para cifrado y descifrado utilizando AES en tubería y secuencial de XCB, HCTR, HCH, HCHfp, EME y TET. Cabe destacar que son las primeras implementaciones de estos modos de operación reportadas en la literatura.
- Se encontraron formas de paralelizar operaciones en los TES para hacerlos más eficientes.
- Se observaron detalles que ayudan a la construcción de TES eficientes para su implementación en hardware.
- Se obtuvo el primer análisis experimental sobre XCB, HCTR, HCH, HCHfp, EME y TET, utilizados para cifrar sectores de un disco. Incluyendo parámetros importantes como área, tiempo, latencia, frecuencia y TPA.
- Se generó un amplio análisis de la eficiencia de cada TES, tanto en recursos de hardware utilizados como en su comportamiento en el tiempo.

7.2. Conclusiones del trabajo realizado

Es importante destacar que en los últimos años se han publicado una gran cantidad de TES, se ha realizado un análisis riguroso sobre su seguridad pero no se habían realizado implementaciones de estos. La aplicación natural de los TES es el cifrado/descifrado de sectores de discos, aplicación la cual en los últimos días ha tomado mucha relevancia. La aplicación de cifrado de discos exige un modo de operación que sea rápido, capaz de cifrar/descifrar datos a la velocidad de 3 Gbps, tasa de transferencia más alta de las tecnologías emergentes S-ATA (acrónimo de *Serial Advanced Technology Attachment* y NCQ (acrónimo de *Native Command Queuing*) [43]. Las implementaciones realizadas en esta tesis a excepción de HCTR que tiene una tasa de cifrado de 3.67 Gbps se quedan cerca de esta meta de velocidad. Utilizar AES c/d en tubería es la mejor opción para esta aplicación, las restricciones de computo no son significativas además de que utilizar AES en forma secuencial tienen un *throghput* muy pobre.

En este trabajo se ha demostrado que la solución propuesta en hardware reconfigurable, significativamente mejora a las mejores soluciones en software existentes en la literatura abierta. Lo anterior se demostró en la tabla 6.7. Además, una aplicación para cifrar un disco trabajaría en alto nivel, i.e, a nivel del sistema operativo. Una aplicación en hardware es totalmente transparente al usuario y trabajaría a nivel del controlador de disco.

Experimentalmente se observaron aspectos de diseño de TES que afectan la eficiencia en su implementación. Si el TES tiene muchos llamados simples al cifrador por bloques que dependan entre sí, el número de ciclos de reloj utilizados se incrementa considerablemente. No es recomendable utilizar varias llaves para el cifrador por bloques, ya que se requerirán varios procesos de generación de llaves de ronda que resultan costosos en tiempo sobre todo si se emplea descifrado. En los esquemas que utilizan función picadillo si se genera la llave para el picadillo con el cifrador por bloques, se tiene la ventaja de utilizar una sola llave de usuario pero existe la desventaja de utilizar un mayor número de ciclos de reloj.

Después del análisis de las implementaciones realizadas se puede decir que los mejores modos de operación para la aplicación de cifrado de discos son HCTR y HCHfp. Aunque EME es más económico en área que HCHfp, existe el problema de que es un modo de operación bajo derechos de patente. En cuestión de seguridad HCTR tienen un límite de seguridad cúbico mientras que el de HCHfp es cuadrático. Con la observación anterior se puede afirmar que HCHfp es el candidato más fuerte de los 6 presentados en esta tesis para ser declarado como estándar.

La aplicación de cifrado de discos se reduce a cifrar/descifrar un mensaje de longitud fija. La mayoría de los TES fueron diseñados para cifrar mensajes de longitud variable, a excepción de EME. En este sentido no es necesario que la salida dependa de la longitud de la entrada. Ahora, si se cifrarán mensajes de longitud variable los mejores esquemas son los picadillo contador picadillo, ya que el cifrado con el modo contador no requiere demasiadas adaptaciones para manejar un bloque incompleto.

En la literatura no se reportan implementaciones de TES en hardware reconfigurable, esta tesis representa el primer análisis experimental acerca de esta familia de modos de operación de cifradores por bloque. Lo anterior nos coloca como pioneros en el área y a la presente tesis como una contribución importante en la búsqueda de un estándar para el cifrado de discos.

7.3. Trabajo futuro

Para la aplicación de cifrado de discos es necesario lograr una velocidad mayor a 3 Gbps, para lograrlo es necesario hacer más eficiente la implementación de AES, para este propósito se pueden utilizar las ideas vertidas en [13]. Teniendo en cuenta que cada vez se van mejorando las implementaciones de AES es posible adquirir nuevas ideas para eficientar la presentada en esta tesis.

Una tarea importante que mejoraría la eficiencia de los TES es adaptarlos específicamente para el cifrado de discos, eliminar la dependencia de la longitud de la entrada. Esto permitiría ahorrar multiplicaciones y llamados al cifrador por bloques. Lo anterior debe hacerse sin afectar el límite de seguridad del esquema.

Un camino alterno para obtener una mayor eficiencia de los diseños presentados en esta tesis es utilizar relojes múltiples, separar el funcionamiento de AES del cálculo de la función picadillo.

Otra labor importante es proponer nuevos TES que sean diseñados considerando ambos caminos, la seguridad y la eficiencia en hardware ya que la aplicación del cifrado de discos es natural para ser implementada en hardware.

Apéndice A

Preliminares Matemáticos

A.1. Grupo Abeliano

Un Grupo Abeliano $\langle G, + \rangle$ consiste de un conjunto G y de una operación definida sobre sus elementos denotada por $+$:

$$+ : G \times G \to G : (a,b) \mapsto a+b$$

Debe cumplir:

- Cerradura: $\forall a,b \in G : a+b \in G$
- Asociatividad: $\forall a,b,c \in G : (a+b)+c = a+(b+c)$
- Conmutatividad: $\forall a,b \in G : a+b = b+a$
- Elemento Neutro: $\exists 0 \in G, \forall a \in G : a+0 = a$
- Elemento Inverso: $\forall a \in G, \exists b \in G : a+b = 0$

A.2. Anillo

Un anillo $\langle R, +, \cdot \rangle$ consiste de un conjunto R con dos operaciones definidas sobre sus elementos denotadas como $+$ y $\cdot$.
Dede cumplir:

- La estructura $\langle R, + \rangle$ es un Grupo Abeliano.
- La operación $\cdot$ es cerrada y asociativa sobre R. Existe un elemento neutral para $\cdot$ en R.
- Las dos operaciones $+$ y $\cdot$ se relacionan por la ley distributiva:

$$\forall a,b,c \in R : (a+b)\cdot = (a \cdot c) + (b \cdot c)$$

El elemento neutro para $\cdot$ se denota por 1. Sí $\cdot$ es conmutativa, el anillo $\langle R, +, \cdot \rangle$ es llamado anillo conmutativo.

A.3. Campo

Una estructura $\langle F, +, \cdot \rangle$ es un campo si satisface dos condiciones:

- $\langle R, +, \cdot \rangle$ es un anillo conmutativo.
- Para cada elemento de F, existe un elemento inverso en F con respecto a la operación $\cdot$, excepto para 0, elemento neutral de $\langle F, + \rangle$.

A.4. Campo finito

Un campo finito o campo de Galois denotado por $GF(q = p^n)$, es un campo con característica p, y q elementos. Un campo finito existe para cada número primo p y un entero n positivo, y contiene un subcampo con p elementos. Este subcampo es conocido como campo base del campo original. Para cada elemento $\alpha \in GF(q) : \alpha \neq 0$, la identidad $\alpha^{q-1} = 1$ es válida.

En criptografia generalmente se utilizan campos con $q = p$ donde p es un número primo y $q = 2^n$, este último es conocido como campo finito de característica 2 o como campo de extensión binaria y se denota como F_{2^m}. Las operaciones en F_{2^m} pueden ser realizadas tanto en su representación canónica como en representación polinomial. Dado que en F_{2^m} sólo se manejan valores de 0 y 1, sus elementos son cadenas de bits lo que facilita su manejo a nivel de hardware.

A.5. Polinomios sobre un Campo

Un polinomio sobre un campo F es una expresión de la forma:

$$b(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + ... + b_2x^2 + b_1x + b_0$$

x es la variable y $b_i \in F$ son los coeficientes.

La potencia más alta de x se denomina grado del polinomio.

A.6. Operaciones sobre polinomios

A continuación se definen la suma y la multiplicación de polinomios.

Suma: Sea $c(x)$ la suma de dos polinomios $a(x)$ y $b(x)$, consiste de la suma los coeficientes con la misma potencia de x, las operaciones se realizan en el campo F:

$$C(x) = a(x) + b(x) \Leftrightarrow c_i = a_i + b_i, 0 \leq i \leq n$$

- Para la suma el elemento neutral es el polinomio con todos sus coeficientes iguales a cero.

- El grado de $c(x)$ es a lo más el grado máximo de $a(x)$ y $b(x)$.
- La suma y la resta son iguales en $F = GF(2^n)$

Multiplicación: Si $m(x)$ es un polinomio irreducible en F, la multiplicación de dos polinomios $a(x)$ y $b(x)$ es su producto algebraico módulo el polinomio $m(x)$:

$$c(x) = a(x) \cdot b(x) \Leftrightarrow c(x) = (a(x) \times b(x)) \ mod \ m(x)$$

La multiplicación de polinomios es asociativa, conmutativa y distributiva con respecto a la suma.

Apéndice B

Tecnología FPGA

Un FPGA (del inglés *Field Programmable Gate Array*) es un dispositivo semiconductor de la familia de dispositivos lógicos programables llamada PLDs (del inglés *Programmable Logic Devices*). Un FPGA contiene decenas de miles de bloques de construcción, conocidos como bloques lógicos configurables (CLBs) con interconexiones programables entre ellos. Estos CLBs pueden ser configurados por los diseñadores para conseguir un nuevo circuito totalmente funcional, virtualmente cualquier tipo de circuito integrado digital puede ser implementado utilizando un FPGA [36].

A raíz que se extendió el uso de los FPGAs para diseño lógico, fueron diseñadas numerosas herramientas de síntesis lógica para ellos. Entre estas herramientas se tienen principalmente los lenguages de descripción de hardware (HDL por sus siglas en inglés) y los editores de diagramas esquemáticos. Los lenguajes de descripción de hardware más utilizados en nuestros días son Verylog y VHDL (*Very High Speed Integrated Circuits Hardware Description Language*).

Una de las ventajas que tienen los FPGA´s es la capacidad reconfigurarlos en tiempo de ejecución, por esta razón se emplean para realizar computo reconfigurable. Los fabricantes más populares de FPGA´s son xilinx [48] y Altera [1], entre ambas ocupan alrededor el 70 % del mercado.

B.1. Familia Virtex4

Para el desarrollo de esta tesis se utilizó un FPGA de la familia Virtex4 [49] de Xilinx. El componente principal de los FPGAs son los CLBs en la figura B.1 se muestra su arquitectura.

Cada elemento de un CLB es conectado a una matriz de direccionamiento para la interconexión entre ellos. Un CLB contiene 4 *slices* que se agrupan en pares, cada par es organizado como una columna. *SLICEM* indica el par de *slices* en la columna izquierda, y *SLICEL* indica el par de *slices* en la columna derecha. Cada par en una columna tiene una cadena de acarreo diferente, sólo los *slices* en *SLICEM* tienen una cadena de acarreo común.

Las herramientas de Xilinx designan los *slices* como sigue: una *X* seguida por un número que identifica una columnas de *slices* y una *Y* seguida de un número que identifica la posición de cada

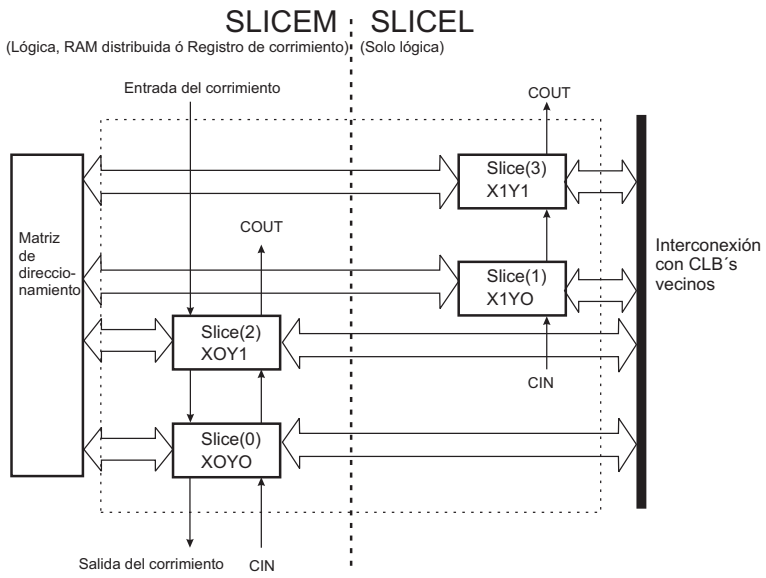


Figura B.1: Arquitectura de un CLB.

slice en un par, puede tomarse como la fila del CLB. Los *slices* $X0Y0$ y $X0Y1$ forma la columna *SLICEM* y el par $X1Y0$ y $X1Y1$ constituyen la columna *SLICEL*. Para cada CLB, *SLICEM* indica el par de *slices* etiquetados con números pares *SLICE(0)* o *SLICE(2)*, y *SLICEL* el par con números impares *SLICE(1)SLICE(3)*.

Los elementos en común en ambos pares de *slices* (*SLICEM* y *SLICEL*) son dos generadores de funciones LUT´s (*look-up tables*)), dos elementos de almacenamiento, amplios multiplexores, lógica aritmética y compuertas aritméticas. Estos elementos son usados por ambos *SLICEM* y *SLICEL* para proporcionar funciones lógicas, aritméticas y de ROM. *SLICEM* soporta dos funciones adicionales: almacenamiento de datos usando RAM distribuida y corrimiento de datos usando registros de 16 bits. Los recursos disponibles en un CLB se enlistan en la tabla B.1.

Slices	**LUTs**	**Flip-flops**	**Multiplicadores ANDs**	**Aritmética y cadenas de acarreo**	**RAM distribuida**(1)	**Registros de corrimiento**(1)
4	8	8	8	2	64 bits	64 bits

Tabla B.1: Recursos lógicos en un CLB.
(1) Sólo en *SLICEM*.

Los FPGAs Virtex4 tienen bloques de memoria RAM de dos puertos empotrados que pueden ser utilizados por el diseñador. Cada bloque de memoria almacena 18 $Kbits$ de datos. La lectura y la escritura son operaciones síncronas, los dos puertos son simétricos e independientes, sólo comparten los datos almacenados. Cualquiera de los dos puertos puede ser configurado con cualquiera de las siguientes relaciones: $16K \times 1$, $8K \times 2$ hasta $512K \times 36$. El contenido de la memoria puede

ser definido o eliminado por la configuración de un flujo de bits.

Específicamente las memorias de doble puerto los datos pueden ser escritos por cada uno o ambos puertos al igual que la lectura. Cada operación de escritura es síncrona, cada puerto tiene su propia dirección, entrada de datos, salida de datos, reloj, activación del reloj y activación de escritura. Cada operación de lectura es síncrona y requiere de un flanco de reloj.

Los bloques *IO* se utilizan para las entradas o salidas al circuito que se diseña. Cada bloque *IO* puede funcionar tanto como entrada como salida ó como un puerto de 3 estados. La cantidad de bloques *IO* varía dependiendo del dispositivo específico.

Los recursos disponibles en el FPGA utilizado en esta tesis se muestran en la tabla B.2.

Arreglo filas$\times$ columnas	**LUTs**	**Slices**	**RAM Distribuida**	**Bloques de RAM**	**Bloques de** *IO*
192×64	110592	49250	768	240	960

Tabla B.2: Recursos lógicos en el FPGA xc4vlx100-11ff1148 de xilinx.

Referencias

[1] Altera. FPGA and structured ASIC: Altera, the leader in programmable logic, 1995. <http://www.altera.com>.

[2] Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX mode of operation. In *FSE*, pages 389–407, 2004.

[3] D. J. Bernstein. A state of art message authentication code. Available in <http://cr.yp.to/mac.html.

[4] Ramesh Karri Bo Yang, Sambit Mishra. A high speed architecture for galois/counter mode of operation (GCM). Cryptology ePrint Archive, Report 2005/146, 2005. `http://eprint.iacr.org/` .

[5] J. Lawrence Carter and Mark Wegman. Universal classes of hash functions (extended abstract). In *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112, New York, NY, USA, 1977. ACM Press.

[6] Debrup Chakraborty and Palash Sarkar. HCH: A new tweakable enciphering scheme using the hash-encrypt-hash approach. In Rana Barua and Tanja Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 287–302. Springer, 2006.

[7] Debrup Chakraborty and Palash Sarkar. A new mode of encryption providing a tweakable strong pseudo-random permutation. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 293–309. Springer, 2006.

[8] Debrup Chakraborty and Palash Sarkar. HCH: A new tweakable enciphering scheme using the hash-counter-hash approach. Cryptology ePrint Archive, Report 2007/028, 2007. `http://eprint.iacr.org/` .

[9] R. Housley D. Whiting and N. Ferguson. Counter with CBC MAC (CCM). Submission to NIST, June 2002, 2002.

[10] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[11] Kristian Gjøsteen. Security notions for disk encryption. In *ESORICS*, pages 455–474, 2005.

[12] Virgil D. Gligor and Pompiliu Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. In *FSE*, pages 92–108, 2001.

[13] Tim Good and Mohammed Benaissa. AES on FPGA from the fastest to the smallest. In *CHES*, pages 427–440, 2005.

[14] Shai Halevi. EME*: Extending EME to handle arbitrary-length messages with associated data. In *INDOCRYPT*, pages 315–327, 2004.

[15] Shai Halevi. Invertible universal hashing and the TET encryption mode. In *CRYPTO*, pages 412–429, 2007.

[16] Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In *Advances in Cryptology - CRYPTO '03*, volume 2729 of *Lecture Notes in Computer Science*, pages 292–304. Springer, 2003.

[17] Shai Halevi and Phillip Rogaway. A parallelizable enciphering mode. In *The RSA conference - Cryptographer's track*, volume 2964 of *Lecture Notes in Computer Science*, pages 292–304. Springer, 2004.

[18] IEEE Security in Storage Working Group (SISWG). PRP modes comparison IEEE p1619.2. IEEE Computer Society, March 2007. Available at:http://siswg.org/.

[19] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-key CBC MAC. In *FSE*, pages 129–153, 2003.

[20] Charanjit S. Jutla. Encryption modes with almost free message integrity. Cryptology ePrint Archive, Report 2000/039, 2000. `http://eprint.iacr.org/` .

[21] Charanjit S. Jutla. Encryption modes with almost free message integrity. In *EUROCRYPT*, pages 529–544, 2001.

[22] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. 7:595–596, 1963.

[23] C. K. Koc and F. Rodríguez Henríquez. On fully parallel karatsuba multipliers for $GF(2^m)$. In *International Conference on Computer Science and Technology CST*, pages 405–410, 2003.

[24] Tadayoshi Kohno and John Viega. The CWC authenticated encryption (associated data) mode. IACR ePrint Archive, 2003.

[25] H. Lipmaa. Fast implementations: Complete AES (rijndael) library, Octubre 2006. Available in <http://home.cyber.ee/helger/implementatons>.

[26] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 31–46, London, UK, 2002. Springer-Verlag.

[27] Mitsuru Matsui. How far can we go on the x64 processors? In *FSE*, pages 341–358, 2006.

[28] Mitsuru Matsui and Junko Nakajima. On the power of bitslice implementation on intel core2 processor. In *CHES*, pages 121–134, 2007.

[29] D. McGrew and J. Viega. The Galois/Counter mode of operation (GCM). Submission to NIST Modes of Operation Process, 2004.

[30] David A. McGrew and Scott R. Fluhrer. The extended codebook (XCB) mode of operation. Cryptology ePrint Archive, Report 2004/278, 2004. http://eprint.iacr.org/ .

[31] David A. McGrew and Scott R. Fluhrer. The security of the extended codebook (XCB) mode of operation. Cryptology ePrint Archive, Report 2007/298, 2007. http://eprint.iacr.org/ .

[32] David A. McGrew and John Viega. Arbitrary block length mode, 2004. http://grouper.ieee.org/groups/1619/email/pdf00005.pdf .

[33] Moni Naor and Omer Reingold. A pseudo-random encryption mode. Manuscript available from http://www.wisdom.weizmann.ac.il/ñaor/. 1997.

[34] Dengguo Feng Peng Wang and Weling Wu. HCTR: A variable-input-length enciphering mode. In Dongdai Lin Dengguo Feng and Moti Yung, editors, *CISC*, volume 3822 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2005.

[35] Rodríguez Henríquez Francisco José Rambó. *New Algorithms and Architectures for Arithmetic in GF*(2^m) *Suitable for Elliptic Curve Cryptography*. PhD thesis, Oregon State University, June 2000.

[36] Francisco Rodríguez-Henríquez, N. A. Saqib, A. Díaz-Pérez, and Cetin Kaya Koc. *Cryptographic Algorithms on Reconfigurable Hardware (Signals and Communication Technology)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[37] Phillip Rogaway. Authenticated-encryption with associated-data. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 98–107, New York, NY, USA, 2002. ACM Press.

[38] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB a block-cipher mode of operation for efficient authenticated encryption. pages 196–205, 2001.

[39] Giacinto Paolo Saggese, Antonino Mazzeo, Nicola Mazzocca, and Antonio G. M. Strollo. An FPGA-Based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm. In *FPL*, pages 292–302, 2003.

[40] Saqib Nazar Abbas. *Efficient Implementation of Cryptographic Algorithms in Reconfigurable Hardware Devices*. PhD thesis, CINVESTAV-IPN, September 2004.

[41] Palash Sarkar. Improving upon the TET mode of operation. Cryptology ePrint Archive, Report 2007/317, 2007. http://eprint.iacr.org/ .

[42] B. Sunar and Ç. K. Koç. Mastrovito multiplier for all trinomials. *IEEE Trans. Comput.*, 48(5):522–527, 1999.

[43] Seagate Tecnology. Internal 3.5-inch (sata) data sheet. Available in <http://www.seagate.com/docs/pdf/datasheet/disc/ds_internal_sata.pdf.

[44] Wade Trappe and Lawrence C. Washington. *Introduction to Cryptography with Coding Theory (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2005.

[45] Shuenn-Shyang Wang and Wan-Sheng Ni. An efficient FPGA implementation of advanced encryption standard algorithm. In *Proceedings of the 2004 International Symposium on Circuits and Systems*, pages 597–600, 2004.

[46] Xilinx. *Block RAM (BRAM) Block v1.00a*, 2004.

[47] Xilinx. *Dual-Port Block Memory Core v6.2*, 2005.

[48] Xilinx. FPGA and CPLD solutions from xilinx, inc, 2006. <http://www.xilinx.com>.

[49] Xilinx. *Virtex4 Family Overview*, 2007.